

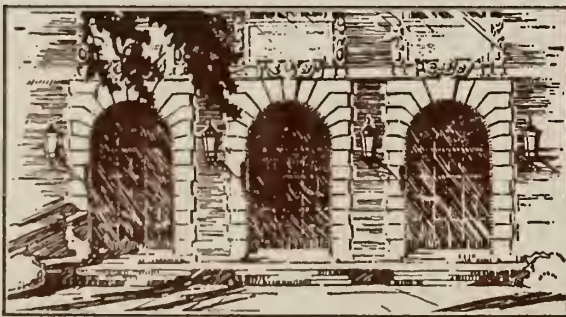
LIBRARY OF THE
UNIVERSITY OF ILLINOIS
AT URBANA-CHAMPAIGN

510.84

Ilwr

no. 463-468

cop. 2





Digitized by the Internet Archive
in 2013

<http://archive.org/details/grasssystemsoftw468mich>

math

162

468

GRASS: System Software Description

by

M. J. Michel

August 1971



THE LIBRARY OF THE

SEP 10 1971

UNIVERSITY OF ILLINOIS
AT URBANA-CHAMPAIGN

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

REPORT NO. 468

GRASS: System Software Description*

by

M. J. Michel

August 1971

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS
URBANA, ILLINOIS 61801

* Supported in part by the Atomic Energy Commission under contract
US AEC AT(11-1)1469.

TABLE OF CONTENTS

	<u>Page</u>
PREFACE	
LIST OF FIGURES	
1. LOCAL SUBSYSTEMS.	1
1.1 <u>General GLASP-Segment-GUTS Linkage</u>	3
1.2 <u>General Data Structuring</u>	4
1.3 <u>Console Vector Detail.</u>	6
1.4 <u>Picture Control Block Detail</u>	10
1.5 <u>GLASP Routines</u>	12
1.6 <u>GUTS Service Routines.</u>	14
1.7 <u>Program Segment Descriptions</u>	26
2. REMOTE SUBSYSTEMS	36
2.1 <u>G8ØPERAT</u>	38
2.2 <u>SPACT.</u>	44
FIGURES	46
LIST OF REFERENCES.	74
APPENDIX	
A. Buffering Control Program (ACID).	76
B. System Generation and IPL	82
C. GLASP Routines--Detail.	89
D. GUTS Service Routines--Detail	101
E. Program Segments.	119
CALR (CALR1):0.	120
GSAM (GSAMV1):1	123
GUT1 (TEXTN):30.	125
GUT2:31	129
GUT3 (REACT):32	133
DEX1 (DEXCHK):33.	138
PSIR (IRINIT):37.	139

PREFACE

An overview of the Graphical Remote Access Support System (GRASS) is given in [1]. Reference [7] describes the operation of the system from a terminal user's viewpoint. Reference [8] presents facilities available to programs, executing under the remote portion of the system, that communicate with the local terminals. All remaining references provide additional background information concerning the design and implementation of the system. Familiarity with the above three references and with PDP-8 and 360 programming is assumed in the document. The system's software (and this discussion) is divided roughly into two sections: local (satellite) and remote subsystems.

The help of the following individuals who aided in the development of portions of the system is gratefully acknowledged: Mr. R. Haskin (PDP-8/I terminal handler), Mr. H. Levin (PDP-8 filing system), and Mr. A. Whaley (360/75 filing system).

LIST OF FIGURES

<u>Figure</u>	<u>Page</u>
1. System Hardware to Software Relationships.	47
2. Allocation of System Control and Work Areas.	48
3. Console Vector Detail.	49
4. Picture Control Block (PCB).	50
5a. File Control Block	51
5b. Data Block Control Entry Detail.	51
6. PCB Control and Data Structure Relationships	52
7. Line Block (No. 0)	53
8. Text Block (No. 1)	53
9. Instance Block (No. 2)	54
10. Terminal Block (No. 4)	55
11. Mnemonic Storage Block (M.S.B.).	56-57
12. Menu Block	57
13. Display Screen Composition Detail.	58
14. Program Segment Allocation	59
15. G8ØPERAT Flow of Control (Part 1 of 2)	60
16. G8ØPERAT/Subtask Linkage (Part 1 of 3)	62
17. SPACT Flow of Control (Part 1 of 7).	65
18. SPACT/User-Program Linkage <u>After</u> S8ENTRY	72
19. SPACT Terminal Control Area Detail	73

1. LOCAL SUBSYSTEMS

As shown in Figure 1, four subsystems plus "user level programs" (hereafter termed "program segments" or "segments") comprise the local section of GRASS.

The multiterminal monitor (GLASP) is responsible for all device interrupts, storage allocation, and scheduling. Certain minor functions such as operator console support and real-time clock display are also performed by GLASP.

The Information Retrieval package (IR) is used to maintain data on the local mass storage medium, currently a disk. This data includes: library file space, segment storage, and transient buffer areas. In addition, a directory and sorted/unsorted block lists are held on the disk. GLASP, GUTS, and the segments can all invoke execution of IR.

GUTS is a large collection of utility routines for handling all device input/output and for manipulating data structure blocks. Routines in GUTS may be invoked by program segments, either directly or via a scheduler in GLASP.

Since the buffering controller (DPU) is currently a PDP-8/I computer, a program to perform terminal handling (ACID) is needed. ACID controls and interleaves display files being sent to the terminals, and regulates the flow of keyboard/X-Y input coming from the terminals. In addition, a dynamic buffer is maintained for each terminal to aid in keyboard text-line preparation.

System response to user input is controlled by the segments. The segments are, in fact, the driving force within GRASS. For example, keyboard or X-Y (joystick) input from a terminal is handled as follows. ACID notifies GLASP that DPU input from a particular terminal is ready. GLASP examines the execution status of the terminal and of the segment associated with that terminal. When the status indicates "accept input" and "not busy," then execution of the associated segment will begin at an entry point corresponding to "DPU input received." Other entry points exist in this segment (and in all segments) for "timer interrupt received,"

"remote (2701) data record received," and "operator console input received." The executing segment now examines the input and makes an appropriate response. This might involve calling GUTS for data structure manipulation or device input/output, invoking IR, or any combination of such activities. The segment may also change its status, such as what input will be acceptable or what entry points may be subsequently used. As soon as the response is completed, the terminal's status returns to "not busy" and new input, if appropriate according to the new status, will cause segment execution to begin again. Hence, for a particular application, all that is needed is an appropriate segment. Change the segment and obtain a text editor, a library inquiry station, an algebraic manipulator, a graphical compiler, a drawing program, etc. The operation of segments currently available is described in [7].

Figure 2 presents the allocation of GLASP, IR, GUTS, and segments with the associated control/work areas in the satellite computer. Each terminal is related to a console vector (Figure 3) and to a picture control block (Figures 4 and 5). The console vector controls the status of the terminal at any time and is the communication medium between GLASP and the segments. The PCB, on the otherhand, controls the status of the data related to the terminal and is manipulated exclusively by the Data Structure Management (DSMN) routines in GUTS. The general format of terminal data and the relationship of PCB pointers to that data is described in Figure 6. Certain standard data blocks are recognized (and expected) by various GUTS routines; these blocks are presented in Figure 7 through Figure 10. Two additional blocks defined for use by the system are shown in Figure 11 (Mnemonic Storage Block) and Figure 12 (Menu Block).

IR is described in [6]. A functional description of ACID is presented in Appendix A. Programs and procedures for generating and loading the system are described in Appendix B, part 1. Critical areas of the system itself are discussed in the following sections.

1.1 General GLASP-Segment-GUTS Linkage

All code is permanently resident except for the segments-- only one segment can be in core at any time. All code is executed with the CPU interrupt facility $\emptyset\text{FF}$; the only time that a device interrupt can occur is when GLASP is looping in an idle state with interrupt $\emptyset\text{N}$.

Two types of GUTS routines are provided: indirectly and directly callable. The first requires use of a device (disk, DPU, 2701, etc.) for performing its service. Since the device might be busy servicing another request, the segment must call the desired routine indirectly through a scheduler ($\text{R}\emptyset\text{UTX}$) in GLASP. Once the request is made, execution will not be resumed in the segment until the service is completed. The second group performs immediate functions; since execution returns directly to the segment, no contention can arise, so the segment may call these routines directly.

The above provides for and requires that all segments be "semi-re-entrant." That is, all temporary storage locations, pointers, subroutine returns, intermediate results, etc., must be in areas associated with a particular terminal when a $\text{R}\emptyset\text{UTX}$ call is made. Entry to $\text{R}\emptyset\text{UTX}$ causes control to be lost, as far as the segment is concerned, for an indefinite amount of time. While awaiting completion of the particular service, other terminals and/or devices may present input for inspection by other segments. These requests will of course be resolved, resulting in changes to temporary areas and to the segment swap area. In fact, when the original service is completed, a fresh copy of the requesting segment may be brought into the segment swap area, thus wiping out all temporaries, subroutine returns, etc., done within the segment itself. On the otherhand, no such precaution is needed for direct calls, since control returns immediately to the segment, and execution cannot be interrupted by a device (interrupt is $\emptyset\text{FF}$).

Naturally, due to the "semi-re-entrant" nature of segments, any segment can service any number of terminals. For example, if segment 7 was a text editor, and terminals 2, 4, and 5 all wanted to use it, segment 7 would be the "associated" segment for those terminals (loc+4 in the

console vector of each would = 7). Unless segment 7 was replaced in the swap area by another segment due to a request from some other terminal, all three terminals (2, 4, and 5) would be serviced continuously, without segment 7 ever being reloaded.

Areas associated with the particular terminal are: console vector and PCB (and, of course, the data blocks). The workspace save area (console vector) and program segment scratch area (PCB) are in general available to arbitrary segments as intermediate storage. Almost all GUTS service routines require some parameters as input. For direct calls, the segment puts the appropriate parameters into the service parameter area of UTBANK and makes the call. For indirect calls, the parameters must be placed in the service parameter area of GLBANK; the segment then calls RØUTX. These GLBANK parameters are automatically stored by RØUTX in the service parameter save area of the console vector (loc+40). In addition, the needed return information is stored (loc+0,+1) as well as the number of the requested service (loc+2). When the desired service is available, RØUTX copies the stored parameters to the service parameter area in UTBANK, then calls the service as indicated (loc+2). The service routine, when finished, returns to GLASP (at ØPDØNE); the service is marked as available and return is made to the segment. However, prior to the return to the segment, the service parameters in the console vector are copied into the GLBANK service parameter area.

1.2 General Data Structuring

All data related to a terminal (other than flags, pointers, counters, etc., in the console vector or PCB) is located in blocks residing in DSBANK. Each block contains one or more "entries," and pointers are kept in the PCB for accessing data within an entry within each block (c.f. Figure 6). In addition, the address, allocated length, and status (in-use/not-in-use, in-core/not-in-core, fixed/not-fixed) of each block is contained in the PCB (c.f. Figure 5). Each terminal can have fourteen blocks associated with it: five system defined (Figures 7-10), five user defined and four "work." Of the work blocks, only block 10 is

generally available to arbitrary segments. Block 11 is used by GLASP during redisplay of the instance block, block 12 is reserved for GLASP, and block 13 is used for maintaining a menu of the user's pictures and mnemonics (Figure 12).

A "picture" is defined as the five standard and five user blocks, plus the file control block (Figure 5a) that describes them. The standard blocks, since GLASP and GUTS know their formats, can be automatically displayed. The user blocks must be explicitly manipulated by a segment for alteration or display, but can have any contents (provided the block/entry format is maintained). For example, a picture created by a simulation and modeling application segment might use two of the user blocks to contain text entries. These text entries would describe the equations, parameters, and variables needed for the network under consideration. Display or alteration of these entries would be performed only by the segment after a request from the user. The network in question would be described by the standard blocks (lines, text, subpicture instances, terminals) and could be displayed or altered automatically.

A "mnemonic" is defined simply as a Mnemonic Storage Block (Figure 11) and is created from the standard blocks of a picture. Only a mnemonic can be used to create entries in the instance block of one picture that represent an instance of another picture. This was deemed necessary in order that picture regeneration not overburden the resources of the satellite computer. All pictures and mnemonics (e.g., the constituent blocks) for each user are stored in IR. Examining or altering a block can

Examining or altering a block can be performed only via the DSMN routines of GUTS. Assume that a picture has already been associated with the terminal; that is, a picture is "current." This means that the FCB for the picture has been read from IR and placed in the PCB. Also, each of the ten blocks used in the picture has been read from IR and its status indicated in the appropriate PCB control entry. Now, the segment calls a routine in DSMN (the loader) to ensure that the desired block is, in fact, in core. The loader examines the appropriate PCB control entry: if the block is in core, the loader sets the PCB control pointers (BLKNUM, BLKØRG, ENTØRG, and PØINTR), marks the control entry to indicate "fixed," and returns to the segment via ØPDØNE. If the

block was not in core, the loader makes an attempt to allocate space in DSBANK. After space is obtained, IR can be called (internally) to read in the block from an IR temporary file. Of course, if space was not obtained, the loader tries to swap-out (e.g. store in IR under a temporary file) as many unfixed blocks as necessary until adequate space is available for loading the desired block. Note that calling the loader can only be done via RØUTX since a device (the disk) may be used during processing. When the block is in core, the loader sets the pointer and control entry as previously mentioned and returns via ØPDØNE.

The segment can operate on a successfully loaded block by calling other DSMN routines. Most of these can be called directly since they only perform immediate functions, such as moving the block pointers, copying data out of the block, or testing data within the block. Others must be called via RØUTX, since they might cause the block to be expanded or moved. For example, adding an entry or inserting data within an entry may cause the initial allocation for the block to be exceeded. In this case, the allocation is automatically increased by the necessary amount, but in order to obtain enough contiguous core area for the expanded block, DSMN might have to rearrange DSBANK slightly. IR must be called and disk accessing must be performed, thus "losing control" for the segment. As soon as the segment is finished with the block, a DSMN routine is called to mark the control entry to indicate "unfixed." This designation means that no Input/Output or other operations are being performed on the block. Hence, the DSMN loader may store the block in an IR temporary if space is needed for other operations.

1.3 Console Vector Detail

Primary control usage of the console vector fields is described below; other manipulations of various fields are documented in segment descriptions or GLASP routine descriptions (see later).

Locations +0 through +13 are for GLASP control:

- +0...+1 contain the segment return information during a RØUTX call.
- +2 contains the number of the GUTS service requested by the segment in a RØUTX call. A zero in this location indicates the terminal (e.g., execution by the associated segment) is "not busy." A positive number means a GUTS service has been requested and is queued; a negative number (-1) indicates that a requested service is in the process of being performed. Both a positive or negative number is considered to indicate that the terminal is "busy."
- +3 contains the input status of the terminal. Zero indicates that the terminal is physically available, but not in use; 4000 means that the terminal is physically unavailable and should not be used. 2000 indicates that the terminal is in use (active) but no device input is pending. Device input pending for an active terminal is specified by
 - 2001:2701
 - 2002:operator
 - 2004:DPU
 - 2010:timer
 Note that 2003 (2701 and Operator) and other combinations are valid conditions.
- +4 contains the number (0-37) of the segment currently associated with the terminal. "Zero" is the logon monitor segment; this segment is automatically called to execution for a terminal when input arrives for an available but inactive terminal (e.g., a "here is" condition).
- +5...+12 contains the logon string identifying the user at the terminal. This is set only by segment zero.
- +13 contains an internal identifier for the user at the terminal. This is set only by segment zero.

Locations +14 through +17 are for a timer element for the terminal:

- +14 contains a negative number representing the duration of an interval. This location is incremented every 16ms, so -1116 would represent 10 seconds. The timed interval is expired when the count reaches zero and no further intervals are required (see +16, +17 below).
- +15 contains a zero or nonzero flag. If zero, expiration of the timed interval is to be ignored. That is, timer input is NOT to be accepted. Hence, 2010 would NOT be set in +3. If nonzero, timer input is to be accepted, so 2010 would be set in +3.
- +16 contains a negative number specifying the number of intervals comprising a "timed interval." Whenever +14 reaches zero, this location is incremented. If the result is not zero, +14 is reset from +17 (see below) and timing continues. If the result is zero, the timed interval has expired, timing ceases, and timer input pending is set in +3, if applicable. For example, to get an interval of 35 seconds, set +14 = -447 (5 sec.), +17 = -1116 (10 sec.), +16 = -4, and +15 to nonzero. A similar result could be obtained by setting +14 = -4021 (35 sec.), +16 = -1, and +15 to nonzero (+17 is not used). Timed intervals can range from 16ms to about 70 hours.
- +17 a negative number to be copied into +14 in the event +16 is nonzero.

Locations +20 through +23 are for device input entry addresses:

- +20 2701 entry address or zero
- +21 Operator entry address or zero
- +22 DPU entry address or zero
- +23 Timer entry address or zero

Whenever +3 indicates input pending, the GLASP scheduler periodically tests +2 for "not busy." As soon as "not busy" is recognized, the associated segment (specified in +4) is loaded and entered at

the corresponding entry address. Of course, if the corresponding entry address was zero at the time the input arrived, +3 would NOT have been set, and the input would have been flushed. Hence, +20 through +23 specify whether to accept or ignore the different possible device inputs.

Locations +24 through +36 are for filtering terminal X-Y and keyboard input:

+24 Screen segment 0 flag

·

+35 Screen segment 9 flag

The display area of each terminal is separated into 10 blocks (Figure 13). Even when DPU input in general is to be accepted (+22 is nonzero), certain input may be considered as spurious. For example, the segment may only want to allow the user to hit (joystick interrupt) in the menu area or return box. Disabling input from all other screen areas avoids the need to load a program segment just to find out that the input is spurious. Hence, if the flag for an area is zero, DPU input pending will NOT be set in +3, even though +22 is nonzero. If the flag for an area is nonzero, a hit in the area will cause input pending to be set. Moreover, the nonzero flag is the entry address used to begin segment execution. Thus, the segment not only gets the X-Y input but immediately "knows" what screen area was hit.

+36 keyboard input entry address or zero (ignore). As above.

Location +37 is for 2701 indicative text-line handling:

+37 line Y coordinate 1344 to 170. Certain 2701 text records are displayed on the terminal immediately. This location serves as a permanent pointer to the next available line on the display surface. The value is decreased by a line height after each use; the value 144 (bottom) causes wraparound to the value 1344 (top).

Locations +40 through +47 are for service parameters:

+40...+47 the parameter needed for GUTS calls are saved here while the request is enqueued by RØUTX.

Locations +50 through +77 are for segment usage:

+50...+77 all locations are available to segments; consult segment descriptions for use. Exceptions are +50 and +51; consult the REGEN description in section 1.7.

1.4 Picture Control Block Detail

Primary usage of PCB fields is described below. Other manipulations of these areas are outlined in the GUTS and segment descriptions.

Locations +0 through +57 are for the File Control Block (FCB):

+0...+2 The six character name of the current picture.

+3 Type character word for a picture (4000).

+4...+5 Zeroes (Spares)

Consult the REACT segment description for additional conventions pertaining to use of +0...+5.

+6 Zero (reserved for protection bits).

+7 Internal identifier of user who created picture.

+10...+57 control entries for the ten constituent blocks of the picture.

Consult Figure 5b for the detail of a control entry.

Locations +60 through +77 are for four accessory block control entries:

+60...+63 work block. Available for use by any segment. For example, the drawing segments use this block for building mnemonic storage blocks, while the REACT segment uses this block for buffering data blocks transmitted via the 2701.

- +64...+67 regeneration block. Used by the automatic display regeneration routines for buffering mnemonic storage blocks during display of the instance block.
- +70...+73 spare block. Currently available for general use.
- +74...+77 menu block. Contains the logged-on user's list of pictures and mnemonics saved by the drawing segments.

Locations +100 through +117 are control fields:

- +100 BLKNUM. On exit from any DSMN routine to a segment, the address of the control entry for the block accessed by that routine is stored here. Conversely, on entry to any DSMN routine from a segment, this location is used to indicate the "current" block being investigated. The appropriate "current" ENTØRG and PØINTR for the block can be fetched from the control entry described by BLKNUM.
- +101 lockout. Used by the DSMN loader to indicate that no lockout (0) or lockout(-1) occurred while attempting to allocate space for a block. This location is set only by the loader, so all subsequently invoked routines (GUTS and/or segments) can test it.
- +102...+103 spares.
- +104...+105 previous ENTØRG and previous PØINTR. DSMN routines that alter ENTØRG or PØINTR first store the old values here. This is sometimes useful while searching down the entries in a block.
- +106 instance block regen. The automatic display regeneration routines save the number of the control entry that points to the instance block being redisplayed in this location.
- +107 spare.
- +110 MNMØD. One of the DSMN routines that alters PØINTR always adds in the contents of this location. This is very useful for scanning down the entries of a block and looking at a particular data word. The normal value of MNMØD is zero. It should always be reset to zero after use with another value.

+111 spare.

+112...+113
Y, X coordinates. The DPU input filtering routine always stores the Y, X coordinates here for accepted joystick input from the terminal.

+114...+115
reference center. The Y, X coordinates of the screen area reference center are stored in these locations. Several DPU routines utilize these. The normal values are 1000, 1000.

+116...+117
spares.

+120...+177
work area (PCWORK). Generally available for segments. Often used to buffer 2701 indicative text-lines before display at the terminal or to buffer text lines from the terminal before transmission over the 2701. Other usage indicated in the segment descriptions.

1.5 GLASP Routines

The monitor is composed of two sets of routines: system functional and segment functional. The former handle all internal control operations needed by the system and are activated either by device interrupts or by entry from GUTS routines or segments. The latter perform some utility function upon entry from a segment.

System:

Interrupt handler (INTHND): trap all device interrupts, link to GUTS device routines, and link to input routines.

2701 input (FR2701): recognize remote input, set 2001 pending.

Operator input (ØPKB): recognize operator command input, set 2002 pending.

DPU input (DPUINR): recognize DPU input, set 2004 pending.

Timer input (CLKQR): recognize timer input, set 2010 pending.

GUTS service requests (RØUTX): enqueue segment service requests, set request pending.

Scheduler (TSTQUE): text input pending and service request pending for each terminal, attempt to start segment if pending found.

Idler (WAIT): test system priority flags, call scheduler, await device interrupts.

GUTS service completion (ØPDØNE): restart segment execution according to console vector data.

Segment completion (RELESE): mark terminal "not busy," safety unfix of all data blocks, call idler.

Segment:

Load new segment (XCTL): change segment associated with terminal, loads new segment and begins execution.

2701 indicative lines (GSMSGa): displays 2701 text-line message on terminal.

Terminal input filtering (DPUDIN): helps filtering spurious terminal input.

Standard block display (GREGEN): automatically regenerates display at terminal for five standard system blocks.

Text messages (GREGPl): helps set messages into display files for output to terminal.

SVPUT: pass a parameter to a particular service parameter location in UTBANK.

FREDPU: clear DPU flags and blank first part of input buffer.

This call MUST ALWAYS be made as soon as the segment is done examining the DPU input.

RETIME: reset timer element for ten minutes.

FREDIM: call FREDPU and RETIME.

RLV1,RLV2,RLV3: call XCTL for return to a previous segment.

MØVEDT: arbitrary movement of data between any pair of banks.

SADCV: add to the ACC the word at +n in the console vector.

SSTCV: store the ACC at +n in the console vector.

SADPC: add to the ACC the word at +n in the PCB.

SSTPC: store the ACC at +n in the PCB.

STØP: reset the console vector beginning at +14 through +37.

STWK: reset the console vector beginning at +50 through +77.

LDWK: reset the page 0 work area (loc 150-177) from the workspace save area in the console vector (+50 through +77).

SVPRLD: reset the page 0 service parameter area in GLBANK or UTBANK from the console vector (+40 through +47).

SVPRSV: reset the console vector service parameter area (+40 through +47) from the page 0 service parameter area of GLBANK or UTBANK.

TØLAC: test a terminal for active/online/offline status.

GLINE: calculate a line number from a menu Y coordinate.

More detailed descriptions of the GLASP routines are presented in Appendix C.

1.6 GUTS Service Routines

The routines comprising GUTS are divided into numbered groups according to function and to system resource monopolized:

1. Data Structure Management (DSMN) Core
2. Information Retrieval (IR) Disk
3. 2701 Send (TØ2701) 2701 out-channel
4. 2701 Receive (FØ2701) 2701 in-channel
5. Display Processing (DPU) DPU channel and buffer
6. Hardcopy Printing (INK) Inktronic printer

The request number specified in a RØUTX call is the group number. Each service group is associated with a flag word and an entry address. When the flag is zero, the service is not busy (idle); hence, another segment request for its function can be accepted. The terminal's console vector address is stored into the flag word (the service is now busy; subsequent requests from other terminals will be enqueued) and execution is begun at the service's entry address in UTBANK. When the service is completed, ØPDØNE will replace the console vector address in the flag word with zero, setting the service "not busy."

DSMN:

RØUTX entry (MNNTRE): SVPRM1 specifies routine to be used. On return to RØUTX, ACC stored in console vector SUPRM8 location as return code. Before the indicated routine is invoked, the

BLKNUM field in the PCB is used to reset the current block: ENTØRG, PØINTR, and BLKØRG are set from the indicated PCB entry. After the indicated routine terminates, the current values of ENTØRG, PØINTR, and BLKNUM are saved in the appropriate PCB area. This provides a means for passing the current block from one routine to the next, even when the routines must be invoked with RØUTX.

Direct call entry (MNDPC): SVPRM1 specifies routine to be used. ACC contents passed to routine and final ACC contents returned to calling program. Current block pointers set and saved as in MNNTRE.

Load data structure block (MNLØDR): make specified block ready for manipulation. If not in core, bring it in from IR; initialize pointers to the first entry in the block and set status in-core/fixed. This block is now considered as the "current block" being manipulated. (Called via RØUTX).

Set status "Unfixed" (MNUNF): current manipulation of the specified block is at an end; mark it as available to be "swapped out" (Called via direct).

Delete a block (MNDEB): if the specified block is in use, mark it as not in use; free any core it may occupy; and delete its storage temporary from IR if one exists. If the current picture is subsequently stored in IR, this block will NOT be part of the picture. (RØUTX).

Start a block (MNIBA): essentially do an MNDEB to clean up, then set the initial size of the block (initial allocation) and call MNLØDR. Core will be allocated, and the block is marked in-use, in-core, fixed, but NOT initialized. This block is now the current block. If the current picture is subsequently stored in IR, this block will NOT be part of the picture. (RØUTX).

Set first data (MNIBD): set the initial block data desired into the current block; the block is marked initialized. A previous MNIBA call is assumed; all other DSMN routines can NOW be called to manipulate the block. Unless an MNDEB call is made, this block will be part of the picture if subsequently saved in IR. (Direct).

- Swap block out (MNSTØR): explicitly store specified block in an IR temporary, thus freeing whatever core it is occupying. This call is NOT usually needed, since MNLØDR automatically swaps out unfixed blocks as the need arises. (RØUTX).
- Save status (MNBSV): the pointers associated with the current block are saved in the appropriate PCB entry. See MNBRS and MNBTB. Note that the block remains marked as fixed and in-core. (Direct).
- Reset status (MNBRS): the pointers stored in the PCB entry of the specified block are used to reset that block as the current block. The specified block is assumed to be fixed and in-core (e.g., a call to MNLØDR, some arbitrary DSMN calls, and an MNBSV call all preceded this call). See MNBSV and MNBTB. (Direct).
- Block-to-block data transfer (MNBTB): data or entire block entries are copied, inserted, or replaced from one block to another. The FRØM location is specified by the pointers in the PCB entry of the specified block (e.g. saved by MNBSV). The TØ location is specified by the pointers of the current block. After some set up, MNBTB calls the DSMN routine specified, usually MNRPL, to actually perform the operation. See MNBSV, MNBRS, and MNRPL. (RØUTX).
- Next entry (MNNXTE): move the pointers of the current block to the next entry in the block. The current values of ENTØRG and PØINTR are first stored in SHDERG and SHDPTR to allow one-level back tracking. Then ENTØRG is updated to the address of the next sequential entry, while PØINTR is updated to the address of the first data word in that entry plus a specified displacement (MNMØD). See MNFPR. (Direct).
- Move pointer (MNFPR): the PØINTR of the current block is reset within the current entry by the positive or negative amount in the ACC. See MNFPRA. (Direct).
- Reset pointer current (MNFPRA): the PØINTR of the current block is reset from its current position in the entry to the beginning of the entry, as in MNNXTE. See MNNXTE. (Direct).

Reset entry (MNPERG): both ENTØRG and PØINTR of the current block are reset as in MNNXTE, but to the specified n-th entry. (Direct).

Reset and locate (MNFPRP): both ENTØRG and PØINTR of the current block are reset as in MNNXTE, but to the first entry in the block. In addition, the address of the block is returned in the ACC. This address is ONLY to be used when calling IR or 2701 services; explicit manipulation of blocks is not allowed. (Direct).

Load word (MNPLØD, MNELØD, MNBLØD): the word in the current block pointed to by PØINTR, ENTØRG, or BLKØRG, respectively, is returned in the ACC. (Direct).

Store word (MNPSET, MNESET, MNBSET): the ACC is stored at the word in the current block pointed to by PØINTR, ENTØRG, or BLKØRG, respectively. (Direct).

Replace data (MNRPL): the specified data replaces data at the current position indicated by ENTØRG and PØINTR in the current block. Depending on parameters supplied, entire entries can be replaced, inserted, or deleted, and data within an entry can be replaced, inserted or deleted. (RØUTX).

Insert new entry (MNNEN): the specified data is inserted into the current block before the current entry indicated by ENTØRG as a new entry. After some set up, MNRPL is called for the actual operation. (RØUTX).

Inset new entry end (MNNNE): exactly as MNNEN, but new entry is at the end of the current block. (RØUTX).

Insert new data (MNINE): exactly as MNRPL, but allows data to be inserted at the end of the current entry. (RØUTX).

Insert new data end (MNILE): exactly as MNINE, but allows data to be inserted at the end of the last entry of the current block. (RØUTX).

Extract (MNEXTC): the specified amount of data is extracted (copied) to the specified location from the current PØINTR location of the current block. (Direct).

Build mnemonic (MNMNEM): a Mnemonic Storage Block is constructed in block 12 from the standard blocks of the current picture. (RØUTX).

Integrate mnemonic (MNMNIN): the Mnemonic Storage Block currently in block 12 is integrated into the instance block (block 2) and terminal block (block 4) of the current picture. (RØUTX).

Complete remote block transmission (MN2TB): set up the current block to receive a 2701 input record. See F02701 and Appendix B. (Direct).

The above routines, along with a large number of other less often used routines, are described in greater detail in Appendix D, Part 1.

IR:

RØUTX entry (IRREQ): initiate an operation in the local filing system.

This is the only call available (or necessary) for segments desiring direct access to the filing package. The parameters needed are:

SVPRM1 = function

SVPRM2 = file type

SVPRM3-5 = 6 character file name

SVPRM6 = core location of data

SVPRM7 = file length (write) or available core (read)

SVPRM8 = owner's ID (protection)

The functions available include:

READ into DSBANK (6100) or into WKBANK (6000)

WRITE from DSBANK (2100) or WKBANK (2000)

DELETE (2200)

A file is uniquely identified by its name plus the owner's ID plus its type. Note that IR directory blocks are accessed by hash coding the ID and the file name; the type is purposely left out of the hash procedure. This ensures that the directory entries for all files that differ only by type will be located close together, thus speeding retrieval of all such "related" files. The following conventions have been established for storing pictures (1), mnemonics (2), block temporaries (3), and menu blocks (4), in IR:

NAME	ID	TYPE
(1) six user-specified characters	from console vector	4000(8)-4012(8)*
(2) six user-specified characters	from console vector	4040(8)-4047(8)**
(3) 0, PCB control entry, 0 address	0	4001(8)-4017(8) ⁺
(4) 0, 0, user's ID	0	4016 ⁺

All other name/type combinations can be used by segments as desired, except for types 4020-4037 and 4050-4077 which are reserved for future system use.

Note, for example, that the picture DIØDE composed of blocks 0, 1, 2, 4, 5, and 6 with mnemonics DIØDE.0 and DIØDE.1 will be stored in nine related IR files. All nine will have the same six character names and user ID, but the types will be 4000, 40001, 4002, 4003, 4005, 4006, 4007, 4040, and 4041. Note, also, that to read or write the information contained in a data structure block, the block must have been loaded (MNLØDR or MNIBA) and its address obtained via MNFPRT.

A complete description of the operation of IR is presented in [6].

* The last two digits correspond to the respective block number, with 00=file control block, 01=block 0,..., 12=block 9.

** The last digit corresponds to the mnemonic designation, e.g., RESIST.6 implies type 4046.

⁺ Same as for *, but file control block is never stored in a temporary; 4013-4016 are for work blocks 10 to 13 (4017 is currently unused since there is no block designated 14).

T02701:

R0UTX entry (T02701): transmit data to the 360/75. The parameters needed are:

SVPRM1 = record type

SVPRM2 = data bank

SVPRM3 = data address

SVPRM4 = data length minus 1 (0-4095 = 1-4096)

Transmission can only take place when the 2701 is not busy, e.g., a read cannot be in progress. Note that to send the information contained in a data structure block, the block must have been loaded (MNL0DR) and its address obtained via MNFPRT. Record types for data structure blocks are identical to IR file types, e.g. 4000-4017 and 4040-4047. Types 7770-7777 are reserved for monitor-to-monitor internal commands; type 0 always means the record is a simple text-line. Type 1 is used by some segments (REACT, etc.) to indicate joystick X, Y data. All other types, except 4020-4037 and 4050-4077, are available for arbitrary segment use. A good example is provided by REACT; see section 1.7, REACT.

The record actually transmitted is:

+0	+1	+2	+3	...	+n
00N0	TYPE	LEN	←	DATA	→

where N is the terminal number and DATA is the information beginning at the location specified by SVPRM3. If $n \geq 128$ (>125 data words), two records are actually sent. The first contains the terminal number, record type, total length, and the first 125 data words; the second record contains just the remaining data words. This procedure facilitates buffering in the 360/75. Note that the 360/75 uses this two-part convention for sending records to the PDP-8.

F02701:

R0UTX entry (F02701): complete a record transmission from 360/75.

Whenever a send to the 360/75 is not in progress, a read-ready status is maintained in the PDP-8. Due to the two-part record convention (see T02701), the PDP-8 must remember whether the input data is the second part of a record or the first part of a new record. In the first case, the data is only read into the PDP-8 after an appropriate segment call to F02701; before making the call, the segment must provide a buffer for the entire record and must move the first part of the record into the buffer. The F02701 call not only completes the read operation, but also allows the system to reset in case the first part of a new record comes along. In addition, a check is made to see if a T02701 call became pending while the read was in progress; if it did, a write operation is started.

In the second case, when the first part of a new record is expected, the system automatically reads the data into a 128 word buffer. The terminal to be notified (via GLASP routine FR2701) can be ascertained from the terminal number in the first word of the record. After the proper console vector is marked, no further action is taken until the segment makes the F02701 call. Note that this is true even if no second part is expected. System processing of further 2701 input must wait until the appropriate segment examines the record in the system buffer, takes action, and notifies the system that the buffer is now "clear" with the F02701 call.

When the segment's execution begins at its "remote (2701) data record received" entry point, the following information is available in UTBANK:

F2701C is the start of the 128 word buffer containing the first part of the record.

F2701T (F2701C+1) contains the record type (all types identical to T02701 usage).

F2701L (F2701C+2) contains the data length -1 (as in T02701).

F2701D (F2701C+3) is the first data word.

I2701P is a flag containing a 1 (second part expected) or

a -1 (no second part since F2701L < 128 (<125 data words)).

From this information, the segment should be able to decide how big the entire record is and what should be done with it. Even if no second part is expected, the F02701 call must be made so that the system realizes that the buffer is clear and 2701 input or output processing can begin again. The system will not try to read a non-existent second part since the I2701P flag is checked by the F02701 routine. The parameters for F02701 are:

SVPRM2 = data bank

SVPRM3 = data address

SVPRM4 = data length (0 means "use preset remainder"; see example below)

A typical use of the 2701 input facility would be the transmission of a data structure block from the 360/75 to replace a block in the current picture of the receiving terminal. The first part of the record is sent by the 360/75 and automatically read into the standard buffer in the PDP-8. The appropriate console vector is marked, and the associated segment begins execution. The segment uses F2701L to calculate the total number of core pages required by the block; MNIBA is then called to establish a block of this size. MN2TB is called next; this routine moves the data from the standard buffer into the beginning of the new block. It then calculates and returns the address within the block at which the second part of the record, if it exists, would start. In addition, it calculates and sets the remaining record length for the second part ("preset remainder"). The segment sets SVPRM2=DSBANK, SVPRM4=0, and SVPRM3=address-returned-by-MN2TB; and F02701 is called. If the second part exists, the read operation transfers the data directly into

the receiving block. Then the system resets for the next record. If the second part does not exist, just the reset occurs. Note that in this example, the segment did not ever check whether the record was composed of one or two parts, since its actions in either event would have been the same.

DPU:

RØUTX entry (DPUENQ): serialize the use of the buffering controller (DPU). The requesting terminal (segment) now has exclusive control of the terminal output buffer and the buffer is initialized. The segment makes direct calls to other DPU routines to construct a display file in the buffer. When the display file is finished (marked complete), it will be sent to ACID for transmission to the specified terminal. When the buffer has been emptied by ACID, control is returned to the segment via ØPDØNE. See Appendix A. If SVPRM8 is not zero, its contents are assumed to be an immediate command, such as "Erase" or "Disable." This one-word display file is sent immediately to ACID, the DPU is released, and control returns to the segment via ØPDØNE. If SVPRM8 is zero, control returns directly to the segment so direct calls can commence. The output buffer is 4000(8) words, long enough to hold the display file for an entire screen of text. If during direct calls this buffer is exceeded, further calls are ignored (e.g., no data is placed in the buffer) until the file is marked complete by the segment. The truncated file is then sent to ACID.

Direct call entry (DPUPDC): the ACC specifies the routine to be used. ENTØRG and PØINTR are reset from the PCB entry indicated by BLKNUM (e.g., current block) as in DSMN processing for those DPU routines that need to examine the current block. However, the final values of ENTØRG and PØINTR are NOT returned to the PCB.

Initial text-line and complete (DPUA): the buffer is initialized, the specified data is inserted as a single text line at a specified Y, X and the file is marked complete. Control is returned to the segment when the buffer is empty (E).

Initial vector and complete (DPUB): as in DPUA, but the display file is a line between two Y, X pairs.

Initial file and complete (DPUC): as in DPUA, but the display file is any arbitrary data supplied by the segment. Also, if the data length specified is zero, the buffer is not initialized. Rather, the current file under construction is marked complete; control then returns to the segment as in DPUA.

Position beam point (DPUDP): a point command moving the beam to the specified Y, X is added to the current display file. The point may be specified as visible or blanked (not visible). Control returns immediately to the segment (I).

Position beam vector (DPUDV, DPUDI): a vector command moving the beam between a pair of specified Y, X points is added to the current display file. If the beam is specified as visible, a line results. (I).

Text-line (DPUE): the specified data is added as a single text line at a specified Y, X to the current display file. (I).

Add small (DPUF): up to five data words are added to the current display file from the SVPRM4-8. (I).

Add arbitrary (DPUG): arbitrary length data as specified is added to the current display file. (I).

Add vectors (DPUH): arbitrary length data consisting of vector commands is added to the current display file, but an enter-vector-mode command is inserted first and an exit-mode sequence is added at the end. (I).

Add text entry (DPUI): all text-lines in the current block entry pointed to by ENTØRG and PØINTR are formatted for display and added to the current display file. (I).

Add text block (DPUJ): all text-lines in the current block entry pointed to by ENTØRG and PØINTR and all text from all remaining entries in the current block are formatted for display and added to the current display file. (I).

Pass text line (DPUK, DPUKA, DPUKB): transmit specified text line to DPU for placement in terminal's line creation buffer. (E).

Add terminal entry (DPULY): format and add specified terminal entry to current display file. (I).

Add terminal block (DPUL): format all terminals and terminal connections in current block (terminal block 4) and add to current display file. (I).

Add line block (DPUNL): format all point groups in current block (line block 0) and add as visible vector commands to current display file. (I).

Add mnemonic instance (DPUM): format lines and text, or parameters model list, or lines only, or text only as specified from current block (mnemonic storage block 12 or 13) and add to current display file. (I).

A complete list of the DPU routines is presented in Appendix D, Part 2.

INK:

RØUTX entry (INKENQ): serialize the use of the local hardcopy printing device. The requesting terminal (segment) now has exclusive control of the printer output buffer and the buffer is initialized. Control is immediately returned to the segment; the segment makes direct calls to INKLK for each complete line (80 characters or less) to be printed.

Print a line or complete (INKLK): the specified data, a packed character string, is unpacked into the print buffer, and printing begins. Control is not returned to the segment until the line is finished. If the data length is specified as being zero, the service is marked not busy (as would be done by ØPDØNE) and control immediately returns to the segment.

SVPRM5 = character count ($\leq 0 \leq 80$)
 SVPRM6 = data address
 SVPRM7 = data bank

1.7 Program Segment Descriptions

GLASP has provision for 32 concurrently available segments; the normal (typical) allocation of "system" and "application" segments is shown in Figure 14. The minimum number of system segments is three (CALR, DEX1, and PSIR); if the number of valid users is fairly large (>256) and if special functions are added to IR, as many as three more segments may be needed (DEX2, DEX3, and SPIR). Three segments containing utility functions (GUT1, GUT2, and GUT3) are available; these utilities may be called by any segment. A very powerful generalized drawing package is provided (GND1, GND2, and GND3) which requires still another three segments. Hence, the number of truly free segments for application use is usually 20. Supporting the Simulation and Modeling application (GSAM) required less than one-half of a single segment; a shortage of application space seems highly unlikely. Moreover, whenever the system is loaded, any group of 20 application segments may be loaded.

The general activity, and some of the possible events, that may take place during execution of the segment GSAM will be briefly described since this segment exemplifies typical applications support. On entry from CALR (the user picked GSAM from the menu), GSAM uses routine STØP to reset the control area of the console vector; this includes the timer element, device input addresses, and the screen segment filter words. GSAM then enqueues the DPU with a RØUTX call and uses GREGP1 as well as DPU direct call routines to build a display file. The new display, containing some messages and a list of three mode options in the menu area, is sent to the terminal with a DPUC call. When the DPU output buffer in the PDP-8 is empty, the DPU service is marked "not busy" and control is returned to GSAM via ØPDØNE. GSAM returns to GLASP via RELESE to await terminal input, specifically a hit on one of the options in the menu.

Several user hits in the draw, prompt, and light button areas as well as a keyboard text-line are ignored by GLASP; the filter words in the console vector, as set by STØP, indicate these inputs are considered invalid. Meanwhile, an indicative text message (type=0) arrives over the 2701; this input is to be accepted. GSAM calls routine GIMSGA to transfer the line to the terminal at the top of the draw area. Finally,

the user hits in the menu area; GLASP stores the Y, X coordinates in the PCB (via PICSEG). DPUDIN is entered and transfers control to the entry in GSAM specified by the menu area filter word. Since only the Y coordinate needs to be investigated and it is already in the PCB, an immediate call to FREDTM is made. The DPU input mechanism is released for more terminal input, and the timer for this terminal is reset to ten minutes.

The menu line hit was CØNSTR, so GSAM puts return data into the console vector level two return and uses XCTL to call the generalized drawing package, GND1. The user interacts in a similar manner with GND1: building a picture, calling IR, directly invoking REACT, etc. Finally, the user returns from GND1 to GSAM, and GSAM re-initializes the console vector and display as it did on the first entry from CALR.

Now the user hits menu line two: SPECIFY. GSAM changes the console vector control area and puts up a new list of options. The sixth option is chosen, indicating that the user wants to add some equations to the picture. This means text editing in entry one of block 6. MNLDØR is called via RØUTX, but the return code is "NOT IN USE"; GSAM calls MNIBA via RØUTX to establish the block, then MNIBD to initialize it with two empty text entries. The text entry editor in GUT1 is now called with an XCTL (return data is first placed in the level 3 return of the console vector) and with parameters indicating editing of entry one, block 6. The user interacts with the text editor and finally returns to GSAM where the console vector and display are reset with the SPECIFY options list. RETURN is hit, so GSAM returns to its main three option mode; RETURN is hit again, so GSAM returns (exits) to CALR.

Functional descriptions of CALR, GNDR(GND1,GND2,GND3), GUT1 (the text entry editor), GUT3(REACT), and GSAM are presented in [7]. PSIR appears in [6]. Detailed information concerning these segments' use of the console vector work area and the PCB, their device input procedures, as well as some very useful subroutines they employ, is found

in the listings (usually the first few pages of each). Segments of particular interest are GND1 and GUT3. A brief resumé of the facilities provided by GUT2, as well as some additional information on GUT3, follows below. Detailed procedures for calling the utilities in GUT2 appear in the listings; LDNMBK is of particular interest in that segment.

REGEN

This routine is invoked normally ONLY by the GREGEN facility in GLASP; it provides a means for regenerating all of the instance mnemonics referenced in an instance block (usually block 2). However, REGEN can be called by segments to display an arbitrarily located instance block (c.f. REACT).

During automatic redisplay of the standard visible blocks (GREGEN), the existence of an initialized block 2 (instance block) causes GREGEN to XCTL to GUT2 at entry RGINST. RGINST assumes that the block has at least one valid instance reference (three entries). Each instance reference in the block is examined sequentially; the instance name and Y, X position are extracted. The specified mnemonic instance block is loaded from IR into work block 13_g by an internal call to LDNMBK (see below). DPUM is then used to create the appropriate display files. RGINST eventually returns to GREGEN, using console vector locations +51 to reset the terminal's current program ID. The terminal's return address was stored by GREGEN in console vector location +50.

LDNMBK

This routine loads a named block from IR into the specified block of the current picture or into a work block; the listing of the routine is quite descriptive and should be examined. LDNMBK can be called on level 3 as a subroutine by any segment; it is also called internally by RGINST during instance block display (above) or by LDNMHP (load new picture, see below).

SVNMBK

This routine saves the specified block (after loading it into core via MNL/DR) in IR with the indicated name. It can be called on level 3 as a subroutine (e.g., to save some block just created) or internally (e.g., by SVNHP for each block in the current picture). The specified block is unfixed before return.

DLNMBK

This routine deletes the indicated named block from IR. It can be called on level 3 as a subroutine or internally (e.g., by DLNHP for each block in the specified picture).

SVNHP

This routine, called on level 3 only, saves the FCB and all initialized blocks of the current picture in IR with the name specified. Internal calls to SVNMBK are made for each block to be saved. The specified name is also inserted into the user's menu block (see below). Before the FCB is saved, the specified name is inserted at its beginning along with a user ID of 0. Hence, IR will use the current logged-on user's ID in the directory entry created.

LDNHP

This routine, called on level 3 only, loads the specified (named) picture from IR and makes it the current picture. First, all blocks of the old current picture are purged from core with MNDEB. Then the FCB of the new picture is loaded from IR. Using the data in the FCB, each initialized block in the new picture is loaded with an internal call to DLNMBK. After being loaded, each block is unfixed.

DLNHP

This routine, called on level 3 only, deletes the named picture from IR. First, the FCB is deleted; then each block is deleted by an internal call to DLNMBK. The picture name is also deleted from the user's menu block (see below).

SVENMN

This routine, called on level 3 only, saves the current contents of block 12₁₀ with the specified name--the block is assumed to be a mnemonic storage block. Before calling SVNMBK internally for the actual IR save, the name is inserted into the block, along with a user ID of 0 (c.f. SVNMBK). The name is also inserted into the user's menu block.

DELNMN

This routine, called on level 3 only, deletes the named mnemonic from IR and the mnemonic name from the user's menu block.

(Note that a routine to load a named mnemonic is NOT provided since this is accomplished merely by a direct call to LDNMBK)

SNDHDR

This routine, called on level 3 or internally by SNDPC (below) transmits the specified block over the 2701 via a RØUTX call to TØ2701.

SNDBLK

This routine, called on level 3 or internally by SNDPC (below), transmits the specified block over the 2701 via a RØUTX call to TØ2701. The indicated block must already be loaded.

SNDPC

This routine, called on level 3 only, transmits the current picture over the 2701. SNDHDR is first called internally to send the FCB; then, after appropriate MNLØDR calls, SNDBLK is internally called for each initialized block.

MENUD

This routine, called on level 3 or internally (by DLNMHP, DELNMN), deletes the indicated name from the user's menu block. The block is first loaded via MNLØDR; then a scan routine (MENUSW) tries to locate the name. If found, the name is removed by a call to MNRPL which compresses the block at that point.

MENUA

This routine, called on level 3 or internally (by SVNMHP, SVENMN), adds the indicated name to the user's menu block. The block is first loaded; then MENUSW tries to locate the name. If not found, the name is inserted by a call to MNRPL which expands the block at the alphabetic location reached by MENUSW.

MENUX

This routine, called on level 3 or internally, extracts the indicated n-th name from the user's menu block. For example, MENUX is usually called after a menu area hit in GND1; it is also used during CALR's purgeout processing.

DIMENN

This routine, called on level 3 or internally, displays the first 32 names in the user's menu block in the menu area. After loading the menu block, DIMENN enqueues the DPU and builds the display file via DPUE calls. The file is sent to the display; DIMENN unfixes the block and returns.

DIMNF1, DIMNF2, DIMNF3

These are three entry points in DIMENN, called on level 3, that cause display of the second, third, and fourth 32-name groups, respectively, in the user's menu block.

REACT(GUT3)

As previously discussed, this program provides generalized user-to-remote-program communication facilities. Two areas bear special mention; the first involves 2701 record handling.

Unless otherwise altered by the program in the 360 (see below), user keyboard lines and joystick hits are sent directly to the 360. Keyboard lines are sent with type=0, joystick hits with type=1. The format is as follows:

Keyboard: 00N0 0000 00nn (packed characters)
 +0 +1 +2 +3... +nn+4

+0 through +2 are the usual record control words, as described previously (N = console number, 0000 = type, nn = length). nn is $\leq 43_8$ since the maximum character line is 72_{10} characters (e.g. $72/2 - 1 = 35 = 43_8$). This is the same form as the so-called "indicative text line messages" sent from the 360 to the PDP-8.

Joystick: 00N0 0001 0002 Y X n
 +0 +1 +2 +3 +4 +5

+0 through +2 as above. +3 and +4 are the hit coordinates, while +5(n) is the screen segment number in which the hit occurred. Note that REACT has a special test to use any hit in the system message area (screen segment 0) as an EXIT REACT request; this cannot be altered by the 360 program.

In addition, as described in [7], REACT can transmit pictures and mnemonics to the 360. The record types for these blocks are identical to the convention used by local IR as presented in section 1.6 (IR). Moreover, LSD uses the same type-word conventions for storing picture and mnemonic blocks in the 360 remote filing system.

Data sent to REACT from the 360 looks much the same. Record type = 0 is an indicative text message line, so the line is displayed on the terminal in the next available location. When the bottom of the draw area is reached, the screen is erased, the frame is redrawn, and the "available location" pointer (kept in +37 of the console vector) is reset to the top of the draw area. Records with the high-order bit set on (e.g. $4xxx$) are assumed to be data structure blocks composing parts of pictures or mnemonics. All other records (types 1-3777) are ignored (flushed). For data structure blocks, type bits 0077 specify the block, 3600 specify options, 4000 must always be on, and 0100 must always be off (else a type of 7770-7777 could be generated which would be confused with a monitor-monitor internal record). The options are listed below; bits are numbered high order (left) to low order (right), 0 to 11.

bit 0: 1 (always).
 bit 1: 1 (replace the similarly numbered block in the current picture with the block in this record).
 0 (put the block in this record into work block 12_8).
 bit 2: 1 (do not display the block in this record).
 0 (display the block in this record immediately on the terminal).
 bit 3: 1 (before displaying block (bit 2=0), erase screen and set available location pointer to draw area top).
 0 (do not erase screen).
 bit 4: 1 (before displaying block (bit2=0), add frame to the screen).
 0 (do not add frame).
 bit 5: 0 (always).
 bit 6-
 bit 11: data block 0 (FCB), 1 (line block 0), etc., or mnemonic as per section 1.6 (IR).

For example, type = 4001 would cause the block in the record to be treated as a line block. It would be put in the work block and a display file would be created from it; this file would be sent to the terminal immediately. A subsequent type 4002 record would cause a text block to be displayed (the old contents of the work block area replaced (lost) by the text block). In both cases, the current picture (FCB plus blocks $0...11_8$) is undisturbed. A type 7004 record would cause the current picture's terminal block to be replaced by the block in the record; no display activity occurs.

Other options can be set with certain words in the FCB; REACT currently uses only the first six words of the FCB. The first four words specify a picture name as usual; if the first word is nonzero, this name is put into the current picture FCB. The fifth word specifies the "auto-save" option:

0 = auto save off (do not save current picture in IR)
 1 = auto save on (save current picture in IR)
 2 = purge (current picture is purged from core)

The sixth word (IMASK in CALLER/COMMUNE) sets the joystick and keyboard filter of the console vector; this word is always saved in the sixth word of the current picture FCB, except when auto-save = 2 (see below).

bit 0: echo (0=display keyboard line typed by the user on the screen at the available location after transmission to the 360; 1=no display).

bit 1-

bit 9: screen areas (1=accept joystick hit in corresponding screen area 1-9; 0=do not accept).

bit 10: keyboard (1=accept; 0=do not accept).

bit 11: (spare).

A typical sequence would be: 1) record type 4000 (FCB) with fifth word = 2 arrives; the current picture is removed (note that when this option occurs, no other options such as reset name or alter filter are executed); 2) record types 7001, 7002, 7003, 7005, and 7006 arrive (in any order), these new blocks (line, text, instance, terminal, and first nonstandard) become the respective blocks of a new current picture; 3) another record type 4600 arrives, the fifth word is NOT=2, so other options are examined. The first word is not zero, so a new name is put in the current FCB; the sixth word is used to reset the console vector filter; and bits 0600 indicate the screen is to be erased and the frame added. However, word five is examined again after the filter is reset and before 0600 is tested. The content is formed to be 1, so the current picture (new FCB with new blocks) is saved in IR; 0600 is not tested in this case. Note that 700x, 460x, and 400x records can arrive at any time; hence, the 360 program can arbitrarily change any block in the current picture (700x). Moreover, the current display can be arbitrarily changed (460x) or added to (400x). This is the essence of the philosophy behind REACT.

Mnemonics can also be saved automatically (but not displayed).

Any block of type >4012 (e.g. 4041) is considered a mnemonic; these blocks must always be marked for "replace in work block" (type bit 1=0). The first six words of the first block entry are examined; if word five=1 (as with an FCB), the assumed mnemonic is saved in IR with the name supplied by the first four words. No other FCB options apply.

The second area of mention is a short note involving the display of an instance block located in work block 12₈. REACT uses a little trick to call GREGNE to accomplish the regeneration. Consult the listing at label "COMINS" since this is a handy procedure to know.

REACT has many interesting sections of code that call system functions (e.g., send picture, send block, save picture, etc.). One of particular interest is the handling of input data structure blocks via MN2TB. Consult the listing at label "COMNTB."

Short, annotated flowcharts giving the overall flow of control for currently available program segments are found in Appendix E.

2. REMOTE SUBSYSTEMS

As is shown by Figure 1, three subsystems plus "user level programs" (hereafter termed "programs") comprise the remote section of GRASS.

The top most monitor level (G8ØPERAT) is responsible for communicating with the operator, for performing the actual 2701 I/O linking the remote and local computers, and for controlling the second level of the monitor. The lower level (SPACT) is a complete multi-terminal package, capable of attaching a separate task (program) to each terminal. Each task can then communicate with its assigned terminal and with the filing system (XFILE). XFILE provides permanent storage for picture and other data generated by programs or transmitted from the satellite PDP-8.

Complete descriptions of the operation of the 2701 link, XFILE, and the monitors, from the viewpoint of the terminal user and his programs, are provided in [8, 9, and 7]. These topics, as such, will not be presented here. Information for generating and running the system is found in Appendix B, Part 2.

G8ØPERAT is considered as the only truly permanent (e.g. continuously resident) part of the system. Current operator commands include:

S name parm	(start second level module called "name" and pass it "parm")
ZAP	(unconditionally stop (e.g. detach) second level)
HALT	(request second level to stop)
DUMP	(output dumps for second level and all its subtasks)
NEVER	(execute STAE as added protection against bomb-out)
UNEVER	(delete STAE condition)
GRASS-ØUT	(terminate execution)

Whenever the second level is NOT attached, any input records from the PDP-8 are considered to be operator commands. In addition, G8ØPERAT can be passed a command as a parameter when it first begins execution. Some special options can be specified in this last case: use of the WTØR's for communicating with the 360 machine operator, as well as use of the system timer, can be suspended.

G8ØPERAT uses the "P8" type macros (see below) for direct communication with the 2701. All hardware error diagnostics and retries are performed on a given read or write operation by G8ØPERAT. No assumptions are made about the data nor is the data transformed in any way (except, of course, in the case of command records). Records are read or written only as requested by the second level of the monitor.

The system's second level can be by any program, and more specifically, might be any program using the "G8" type macros (see below) to request G8ØPERAT to read or write over the 2701. In particular, the second level currently used is the multi-terminal package, SPACT. SPACT has several functions, the primary being the attaching or detaching of any program module associated with any given user terminal in the system. SPACT also maintains an input/output record queuing system to allow each program in the 360 to communicate with the terminal to which it is attached. Any 2701 input record of type=0 (c.f. G8-macro descriptions) with first character of "!" is assumed to be a SPACT command. All other 2701 input is assumed to be data for the attached user programs. SPACT only examines the data portion of command records. The data of all other records is solely the responsibility of the issuing or receiving program module. The format of 2701 records is adequately described in [9]; long data records are handled with a two-part transfer as discussed in FØ2701 (section 1.6).

Any user program can run under SPACT, but again, those of particular interest are the ones that communicate with the terminals via the 2701. Examples are LSD and the various modules of the Simulation and Modeling System [5, 8]. These modules utilize the S8-macros [9] to fetch

records out of SPACT's input queue or place records into its output queue. Since the relationship of program-to-terminal is maintained by SPACT, any program can be attached to any terminal. Furthermore, the S8-macros can be used in re-entrant programs, so if more than one terminal wants the same program simultaneously, only one copy of the program need be in core. For similar reasons, re-entrant user program linkage to XFILE is also possible [9].

In the following discussions, some knowledge of ØS/360 programming has been assumed. Note that the entire monitor and XFILE complex runs exclusively in bulk core; appropriate use of HIARCHY=1 has been made throughout (GETMAIN, ATTACH, LINK, LØAD, XCTL, etc.). The monitors and XFILE require approximately 18-20K bytes of memory; the number and size of user programs to be run concurrently under SPACT dictates the total memory space (REGIØN) that must be assigned to the job.

2.1 G8ØPERAT

The overall flow of control for this module is shown in Figure 15. Salient features include: (numbers refer to indicated flowchart boxes)

1. All G8ØPERAT flags are set to 0 (off). The PARM field from the ØS EXEC statement is examined for initial parameters. The 2701 link is made available via ØPEN and ENBATN (enable attention) macro calls.

Note that ENBATN is a simple installation ØS modification that allows a program to realize that a device desires service. Such a facility is not normally available in ØS 360 but is usually added at installations where timesharing systems are hung on 360 machines.

The parameter string format for the PARM field is:

[[option][command]]

Valid "options" are NM, MS, NT, TM, and {NM,MS}{NT,TM}; an example of the last is MSNT. Unless overridden, MS and TM are assumed by G8ØPERAT. The

"command" is always passed to the operator command analyzer as if it had been received from the 360 operator. This is a handy debugging aid and is also convenient for starting SPACT (e.g., command = S SPACT) automatically. Any G8ØPERAT command can appear in "command."

NM: do not issue WTØR's to operator

MS: issue WTØR's, as normal

NT: do not use system timer

TM: use system timer, as normal

2. Await an event or system timer. An O\$ (and HASP) feature exists that causes termination of a job if that job remains in the wait state (inactive) for more than a certain number of minutes. Since this is an interactive monitor, the possibility of a long wait between events exists. Hence, the system timer is used to interrupt G8ØPERAT (make it active) periodically to avoid cancellation. During debugging, use of the timer in this manner is often not desired, hence the NT option.

3. Execute the indicated command. All commands have the form Verb[Object][parameters] where Verb is the field tested by the command analyzer. It is the responsibility of the routine that executes the Verb to test and use the remaining fields as needed. ZAP, HALT, DUMP, NEVER, UNEVER, and GRASS-ØUT as listed previously are straightforward and self-explanatory. Only "S name parm" (start second level) need be discussed here.

G8ØPERAT uses an ATTACH macro to begin execution of module "name" as a subtask. This subtask is passed a parameter list address in register 1; the list format is:

```
+0  0
+4  A(request ECB's)
+8  A(parm string)
```

The subtask must PØST the first location in the list when it begins executing (c.f. G8ENTRY macro description). This notifies G8ØPERAT that

the subtask has indeed begun executing; G8ØPERAT will then continue normal processing. If ØS 360 PØST's the task-termination ECB specified in the ATTACH macro before the subtask PØST's its location, G8ØPERAT assumes execution of the subtask never started (module not found, insufficient core, etc.).

When G8ENTRY, in the subtask, PØST's list location +0, it also inserts the address of a subtask ECB into the location. If G8ØPERAT PØST's this ECB, the subtask must terminate processing; this ECB is tested whenever the subtask issues a G8WAIT macro. A HALT command, or a GRASS-ØUT command while a subtask is executing, will cause G8ØPERAT to PØST this subtask ECB.

The subtask G8ENTRY macro also extracts the information at +4 and +8 in the list. Whenever the subtask desires 2701 input or output, the appropriate ECB in the pair pointed to by the address at +4 must be PØSTed. The subtask inserts the address of a record request block into the PØSTed ECB (c.f. G8-macro descriptions). The "parm" field in the start command is copied into a 34 byte area before the ATTACH is issued; the address of this area is placed in +8 of the list. Access to the parameter is thus provided; if no parameter was specified, "\$N" is placed in the area.

Figure 16 summarizes these linkages between G8ØPERAT and its subtask.

4. Indicate 2701 operation complete. G8ØPERAT uses the subtask record request block address found in the appropriate read or write request ECB. The first word at this address is PØSTed by G8ØPERAT with a completion code. Consult Figure 16.

5. Execute 2701 operation. The P8-macros, described below, are used to transmit data to the PDP-8. The G8-macros used by subtasks are also described here to increase understanding of their interaction with G8ØPERAT. Consult Figure 16.

Necessary information is extracted by G8ØPERAT from the indicated record request block; P8RETRY is used to perform the desired read or write operation. The actual transmission routines are generated by P8ØPRS and P8ØPRSX. Lower level P8-macros (P8RCCW, P8WCCW, and P8WAIT) are available for more flexible use of P8ØPRS facilities.

label P8RETRY ØP, data addr, len, retry count [,ECBLST=, ØTHER=]

Generates:

```
BAL 1, *+20
DC  A(data addr)
DC  AL1(retry count)
DC  AL2(ØP)
DC  AL2(len)
DC  A(ecb list)
DC  A(other event routine)
BAL 14, PDP8RETRY
```

(R1 points to params, R14 is the return register)

ØP is X'11' (WRITE), X'12' (READ), or (R) (X'11' or X'12' is in register R). Retry count should be less than 20.

ECBLST and ØTHER are optional:

ECBLST is the address of a list of ecb addresses containing at least 'PDP8ECB' BUT NOT 'PDP8ATTN'.

If omitted, a list containing only 'PDP8ECB' is used.

ØTHER is the address of a routine to be entered if an ecb is posted besides 'PDP8ECB' (i.e. ECBLST=lab was coded).

```
R1  = A(ecblist)
R14 = return address
R15 = routine address
```

The routine must "take note," clear the appropriate ecb, and return via R14 to await I/O quiescence. The routine must save registers 0, 1, 13, 14, 15.

On return from P8RETRY, the operation (including any needed retries) is complete and R15=ØK(0), ERR(4), TØ(8), or ATTN(write only) (12).

Note that on initial entry to PDP8RTRY for READ, the system assumes 'PDP8ATTN' has already been posted and then cleared by caller. If retries are needed, the system waits for a new posting of 'PDP8ATTN' before each retry. If such a post does NOT occur within 10 seconds, retry processing is terminated and R15=8(TØ) is returned to the calling program.

P8ØPRSX

(contains the actual retry routines and linkages to P8ØPRS)

P8ØPRS

(contains the actual 2701 I/O routines, DCB, IØB, ECB's, etc.)

Generates:

```
PDP8READ  (start read operation)
PDP8RITE  (start write operation)
PDP8EXCP  (start EXCP I/O as indicated by PDP8READ or PDP8RITE)
PDP8WAIT  (await EXCP completion)
PDP8DCB   (link to 2701)
PDP8ELST  (default ECB list for PDP8WAIT)
PDP8ECB   (EXCP completion ECB)
PDP8ATTN  (2701 attention ECB posted by ØS)
```

The user (e.g., G8ØPERAT) must issue

```
ØPEN  (PDP8DCB,(INPUT))
ENBATN DCB=PDP8DCB,ECB=PDP8ATTN
```

before using the 2701 facilities and must issue

```
ENBATN DCB=PDP8DCB
CLØSE  (PDP8DCB)
```

before exiting to ØS when the job terminates. Also, a DD card named 'PDP8DD' must be included in the job's JCL to define the 2701:

```
//PDP8DD DD UNIT=021
```

Note that if a delay in dispatching the task occurs, it is possible that 'PDP8ATTN' or some other ecb may have been posted between the time 'PDP8ECB' was posted (Operation complete) and the time control is returned to the calling program. Hence, care should be exercised in the use of any "safety" clears of ecb's.

label { P8RCCW
 P8WCCW } data addr, len

Generates:

BAL 14, PDP8READ
BAL 14, or PDP8RITE

R0 contains the byte len: if >4095, this must be specified as in a register, i.e., 'data addr,(R)'.
R1 contains the data addr.

The above call results in the generation of a simple CCW; this CCW is then passed to PDP8EXCP to perform the actual operation. P8RCCW and P8WCCW are used in conjunction with P8WAIT to give the program complete control over the I/O operation. P8ØPRSX actually uses P8RCCW, P8WCCW, and P8WAIT. Note that performing complex I/O functions is possible. Instead of using P8RCCW or P8WCCW, the program can build its own chain of CCWS. Then the program executes BAL 14,PDP8EXCP with the address of the CCW chain in register 1.

label P8WAIT ecbaddrlist,ØKaddr,ERRaddr,TØaddr[,NØT270lecbaddr]

Generates:

BAL	14,	<u>PDP8WAIT</u>	
B	ØKaddr		successful completion return
B	ERRaddr		hardware error return
B	TØaddr		timeout return
[B	NØT270lecbaddr		return if event other than 2701 occurs]

R1 contains the address of the ecb-address list: this list must contain at least PDP8ECB. If only PDP8ECB is included, 'NØT270lecbaddr' should be omitted. PDP8ELST may be used for 'ecbaddrlist' in which case only PDP8ECB in a standard list will be used. In the event of a 'NØT270lecbaddr' return, the program must "take note," clear the appropriate ecb, and issue a new P8WAIT.

G8ENTRY termecb

Generates:

G8RWCØM (request read or write from G8ØPERAT)
G8WTCØM (await operation completion)
G8RDECB (record request block 1 (READ))
G8RTECB (record request block 2 (WRITE))

This macro must be issued by the subtask of G8ØPERAT after addressing is established but before register 1 is destroyed. Besides linking the subtask to G8ØPERAT, G8ENTRY contains the routines used by G8READ, G8WRITE, and G8WAIT for communicating 2701 records via G8ØPERAT. The subtask must provide a word to be used as the HALT request ecb at 'termecb'.

```
label      {G8READ  } data addr, len
           {G8WRITE }
```

Generates:

```
BAL  1, *+12
DC    A(data addr)
DC    X'00', AL1(ØP), AL2 (len)
BAL  14, G8RWCOM
```

```
label      G8WAIT ecb list addr [, ØTHER=routine addr]
```

Generates:

```
BAL  14, G8WTCØM
```

R0 contains 0 or the routine address of an 'other-event' routine, R1 contains addr of ecb addr list (minimally: G8RDECB and G8RTECB; usually plus terminal, etc.) R14 is return register.

On return, R15=ØK(0), ERR(4), TØ(8), or ATTN(write only) (12). On 'other-event' routine entry, the program must take note, clear the ecb, and issue a new wait as in P8WAIT.

2.2 SPACT

The overall flow of control for this module is shown in Figure 17. Salient features include: (numbers refer to indicated flowchart boxes)

1. Set-up to begin. G8ENTRY is called to link SPACT to G8ØPERAT; XFILE is LØAded and its entry address saved. SPACT then sends a monitor-monitor record to the PDP-8; this record (type=7770) indicates to the PDP-8 that the 360 is available.

2. PØST G8ØPERAT with read request. Since a PØST is used, care must be taken that a second PØST does not occur before G8ØPERAT clears the first. Three special tests are thus performed before SPACT issues its outstanding G8READ request.

3. Await event including 2701 Read/Write. The test for part 1 or part 2 of the record that appears in the flowcharts at B and C is really taken care of by the code after the G8WAIT. Whenever a read or write is started, the initializing routine sets the appropriate return address into WTRET or RDRET.

4. Execute the indicated command. After removing the leading "!", SPACT commands are identical in format to G8ØPERAT commands. All are self-explanatory except for Start, which will be discussed briefly here. Start works exactly the same in SPACT as it does in G8ØPERAT, except that the parameter list is different and S8ENTRY performs the linkage from user programs to SPACT. Consult [9]. The linkage performed by S8ENTRY is shown in Figure 18; the terminal control area for each terminal is shown in Figure 19.

As general information, note that the S8-macros interact with SPACT in a manner directly analogous to the interaction of the G8-macros with G8ØPERAT. Furthermore, as is obvious from Figure 17, SPACT transmits long records in two parts in a manner directly analogous to the operation of TØ2701 (section 1.6).

FIGURES

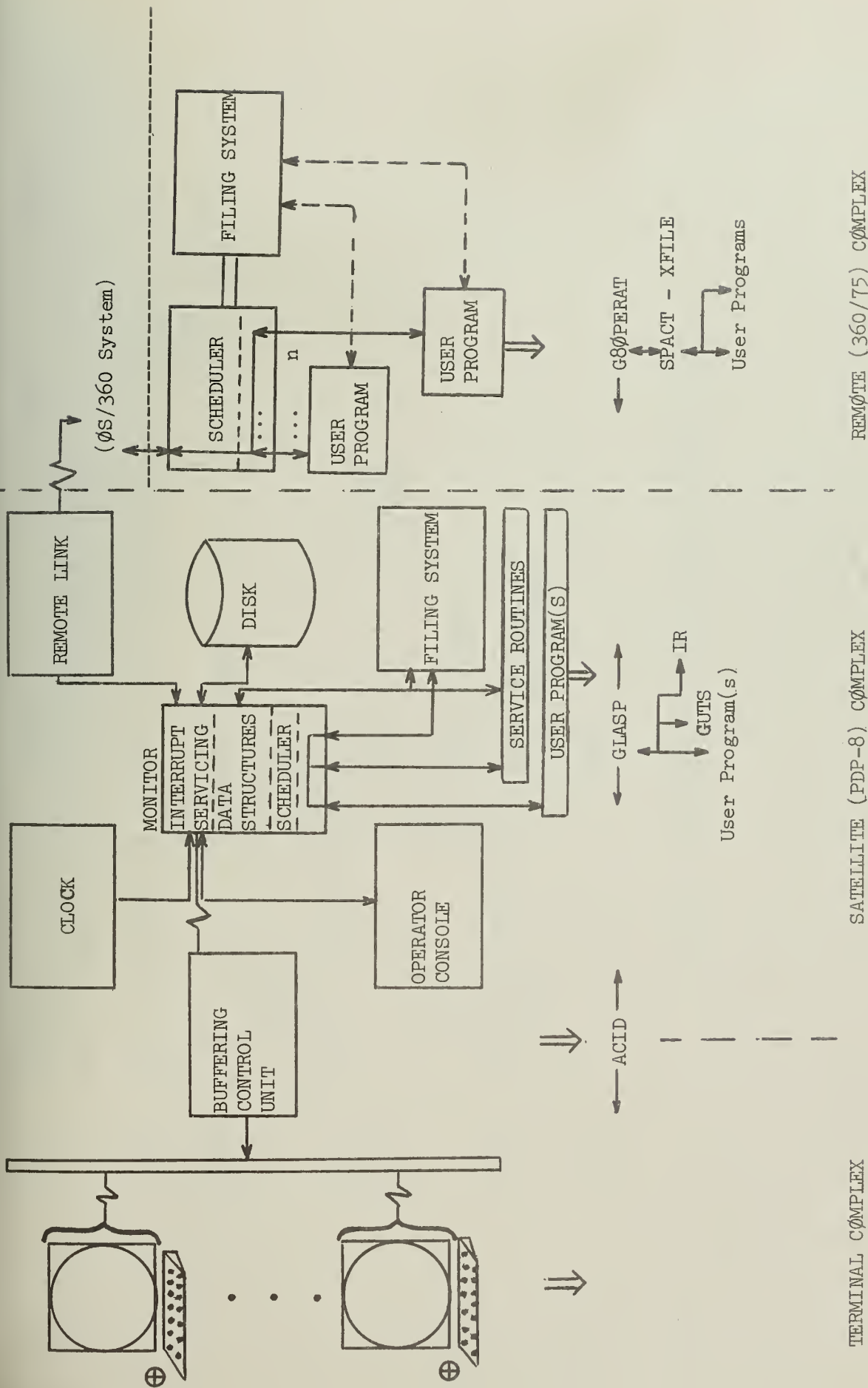


Figure 1. System Hardware to Software Relationships

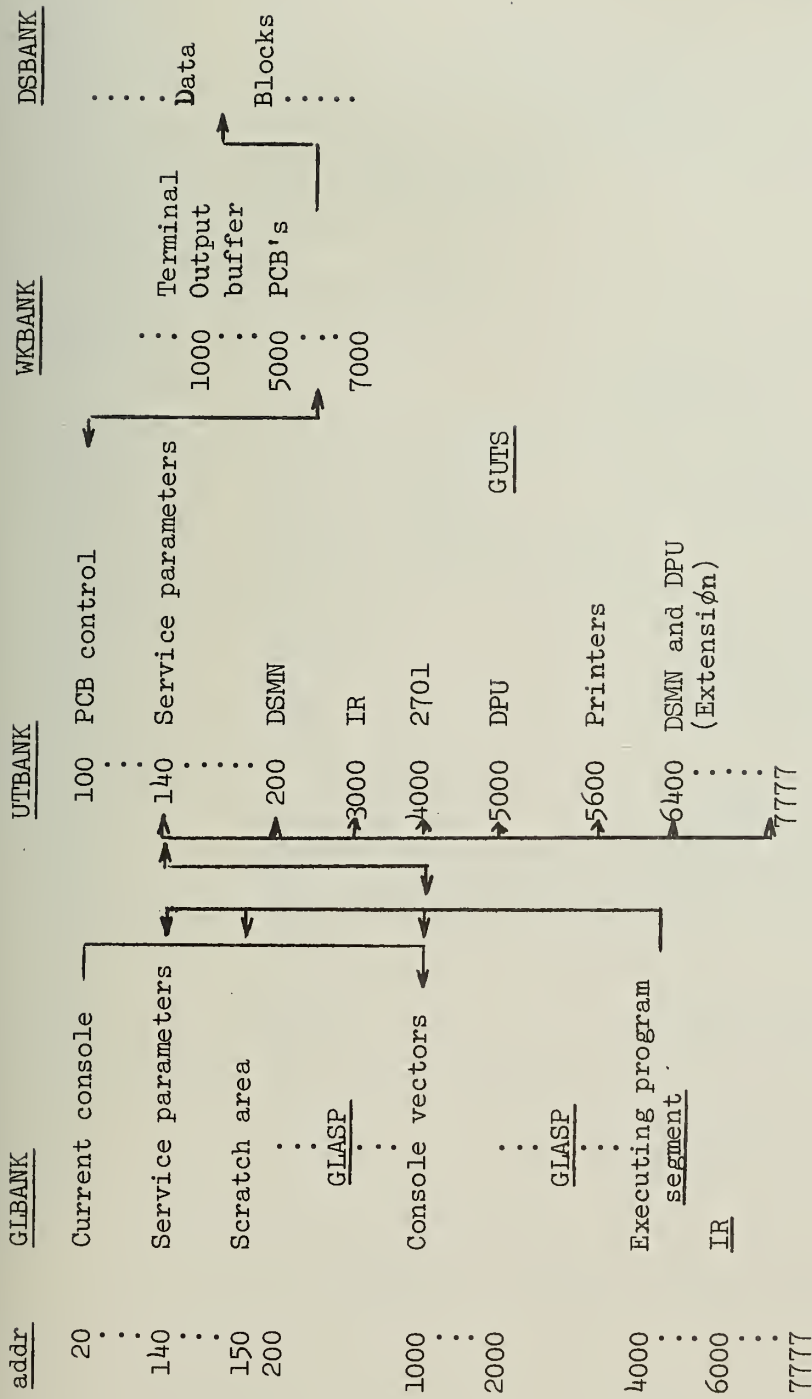


Figure 2. Allocation of System Control and Work Areas

<u>addr</u>	<u>contents</u>	<u>function</u>	<u>initially</u>
+0	CIF CDF	} service request return data	0
+1	ADDR		0
+2	Request No. (0=free(No Request), >0=Service Requested -1=Service Being Processed)		0
+3	Console Status (0=inactive,4000=offline,2000=active, >2000=console input pending)		0
	Input flags: 010 000 00A BCD (A-D=timer, DPU, operator, 2701, respectively)		
+4	Prog ID (0-31)		0
+5	} 'PSNONAMEABCD'		-
:			:
:			:
+12			:
+13	USER ID	(8 bits-Acct No,4 bits- User No.)	-
<hr/>			
+14	Interval Duration	} Timer Queue Element	-
+15	Zero or nonzero		0
+16	Interval Count		-
+17	Interval Duration Reset		-
<hr/>			
+20	2701	} Console Input Prog Entry	-
+21	Operator		-
+22	DPU		Addresses (zero on entry
+23	Timer		from caller)
+24	Screen Seg #0	} Joystick Input Filering Control	-
:			:
:			:
+35	Screen Seg #9		-
+36	Keyboard input control (routine address or 0)		-
+37	2701 Indicative message control (next screen Y)		-
<hr/>			
+40	SVPRM1	} Service Parameter Save Area During GUTS Call	-
:			:
:			:
+47	SVPRM8		-
<hr/>			
+50		} Workspace Save Area	-
:			:
:			:
+77			-

Figure 3. Console Vector Detail

<u>addr</u>	<u>contents</u>	<u>function</u>
+0	Current file	Data block allocation
⋮	Control block (FCB)	
+57		
+60(10)	Work blk	Mnemonic construction, commune temporary, etc.
⋮		
+64(11)	Regen blk	Work area for instance regeneration
⋮		
+70(12)	Spare	
⋮		
+74(13)	Menu blk	Local file names control
⋮		
+100	BLKNUM	Address of last blk investigated
+101	(lockout)	Special lockout flag (-1)
+102	Spare	
+103	Spare	
+104	SHDPERG	ENTORG before last reset
+105	SHDPTR	PØINTR before last reset
+106	SHDRGN	Blk no of instance block during regen
+107	Spare	
+110	MNMØD(0)	PØINTR displacement (idle value = 0)
+111	Spare	
+112	YH	Y position on J.S. hit
+113	XH	X position on J.S. hit
+114	Y scan (1000)	Draw area
+115	X scan (1000)	reference center
+116	Spare	
+117	Spare	
+120	PCWØRK	Program segment scratch area
⋮		(40 words = 80 characters
⋮		8 words = control)
+177		

Figure 4. Picture Control Block (PCB)

<u>addr</u>	<u>contents</u>
+0	Six character name plus two spare characters
+4	Spare (0)
+5	Spare (0)
+6	Protection
+7	User ID
+10(0)	Local lines
+14(1)	Comment text
+20(2)	Subpicture instances
+24(3)	(Reserved)
+30(4)	Terminal Types, Connections, and Terminals
+34(5)	User 1
⋮	⋮
+54(9)	User 5
⋮	⋮
+57-End	

} System standard data
 } block control entries

} User defined data
 } block control entries

Figure 5a. File Control Block

Word 1

CIO 000 ONN NNN

C=0(incore) or 1(not)
 I=0(initialized) or 1(not)
 NNNNN=**number** blocks allocated or 0

Word 2

FUA AAA AOO 000

F=0(not fixed) or 1(fixed)
 U=(reserved)
 AAAAA=block addr in DSBANK

Words 3

ENTØRG last used for this block; saved here on return from any DSMN routine.

Word 4

PØINTR last used for this block; saved here on return from any DSMN routine.

Figure 5b. Data Block Control Entry Detail

PCB supplied control items:

BLKNUM=j (block under consideration)
 BLKORG=A (starting addr)
 ENTORG=B (entry in block under consideration)
 POINTR=C (data in entry under consideration)

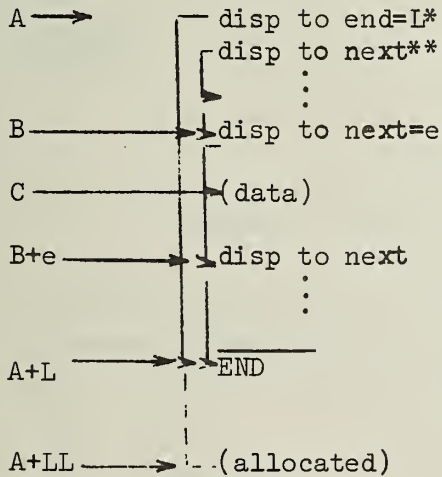
block j control entry:

allocated blocks (NNNNN)=LL
 core addr (AAAAA)=A

control

data

DS BANK data block:



* Disp-to-end ALWAYS < 2048 (i.e., positive); hence, maximum len of data in a block < 2047.

** Disp-to-next-entry ALWAYS < 2047 (i.e., < L); hence, maximum len of data in an entry < 2046.

Figure 6. PCB Control and Data Structure Relationships

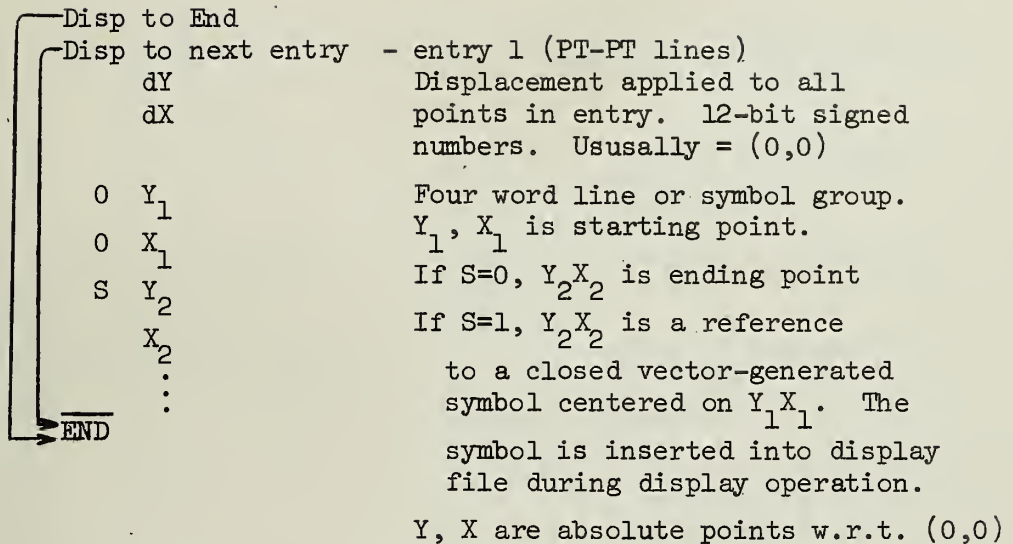


Figure 7. Line Block (No. 0)

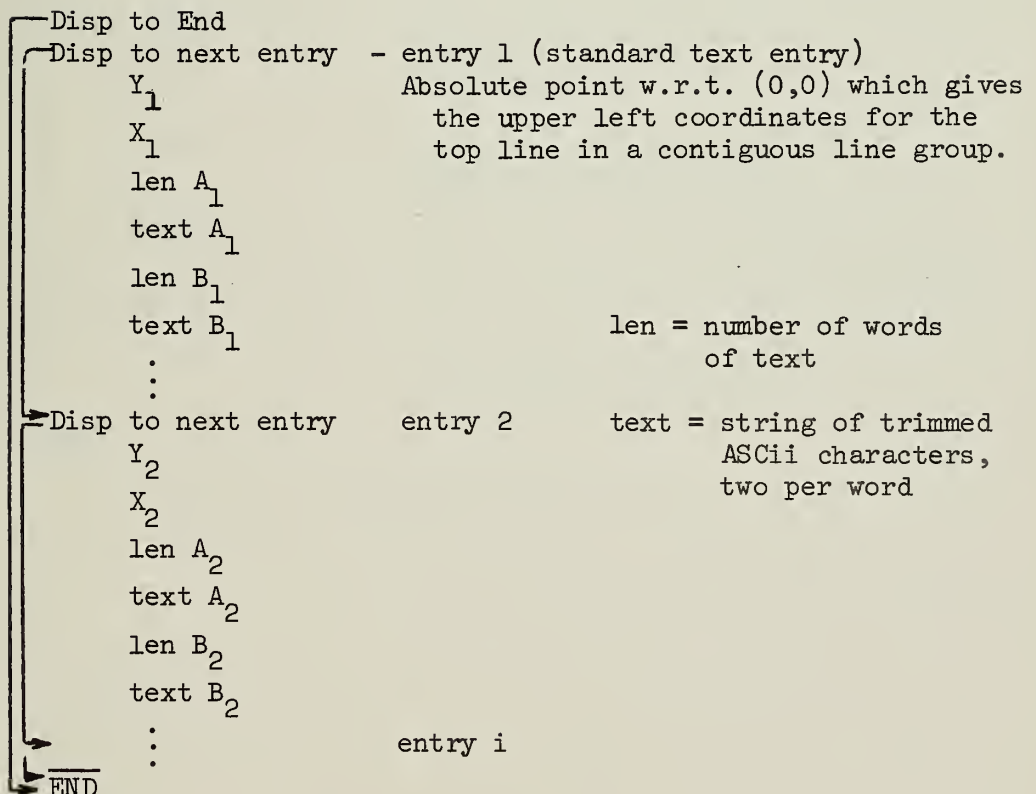


Figure 8. Text Block (No. 1)

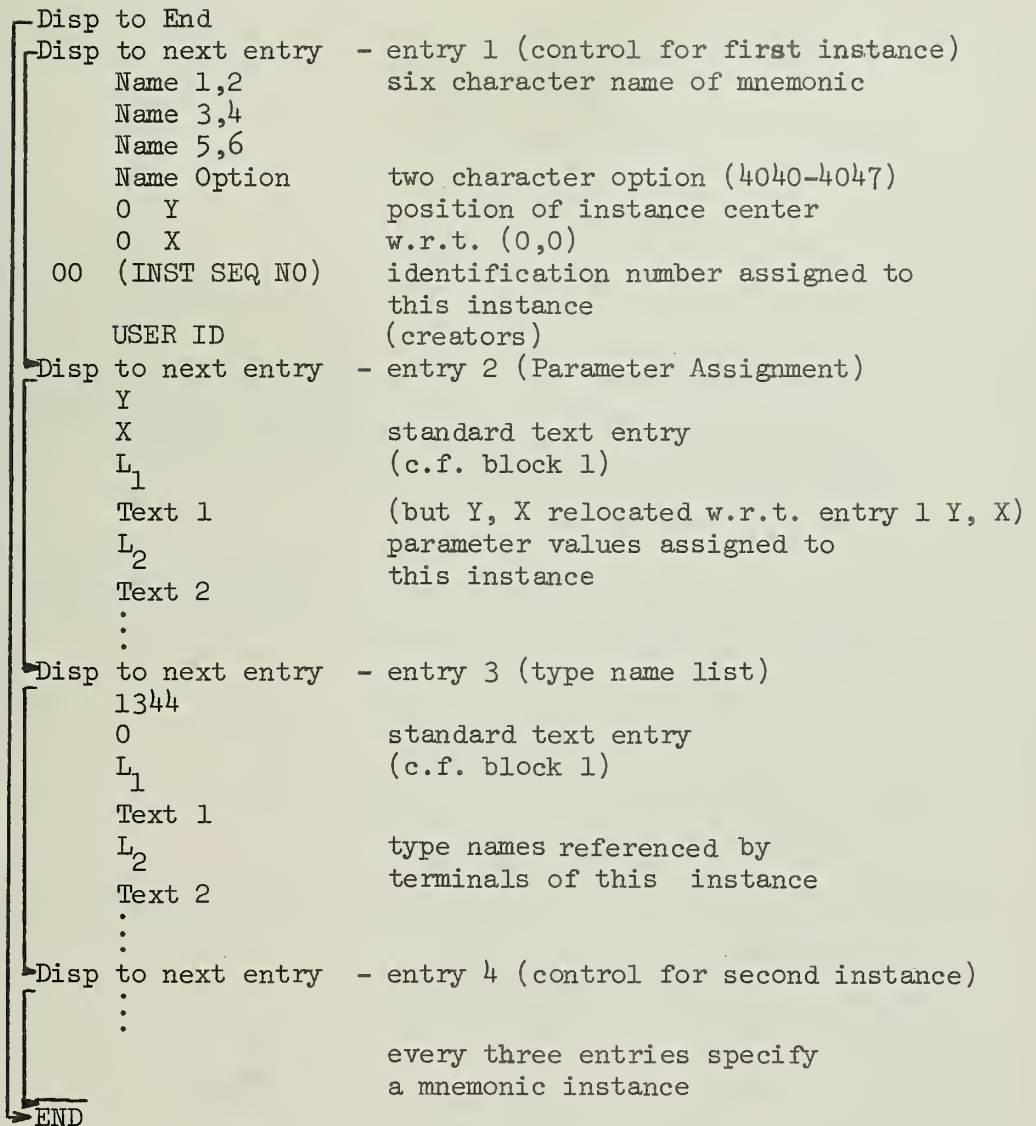
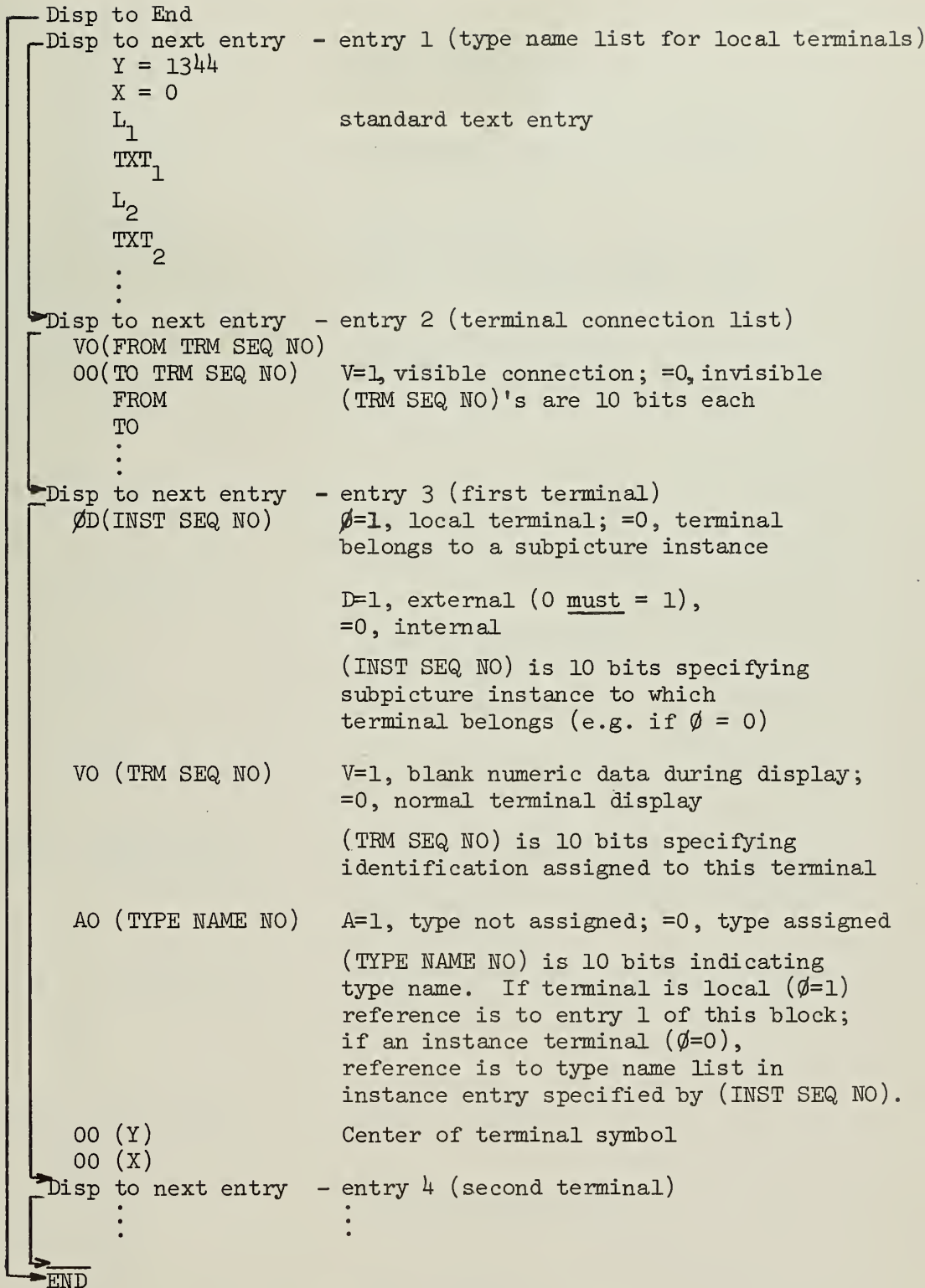


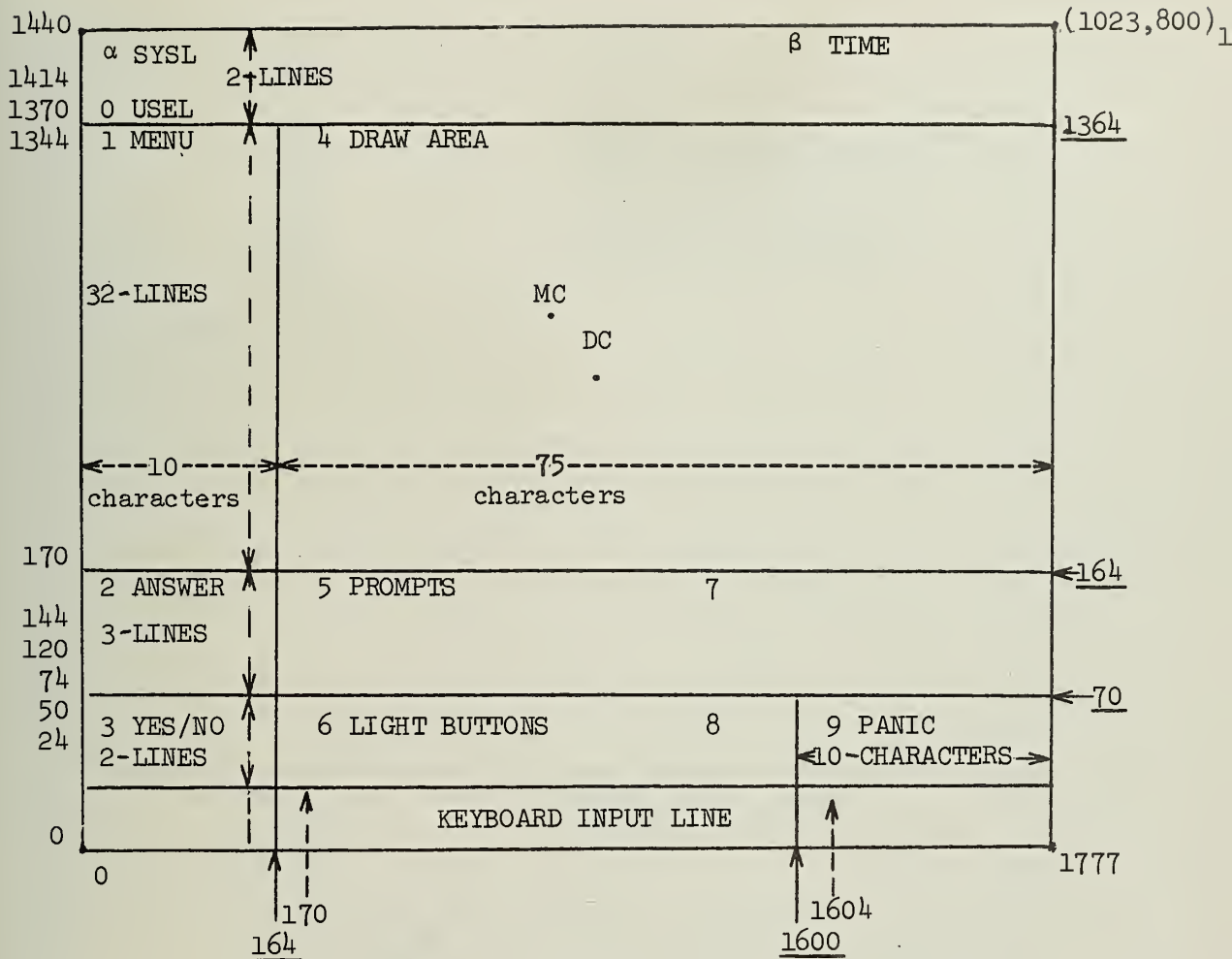
Figure 9. Instance Block (No. 2)



Typical instance/internal terminal appears as . bb37 where 'b37' is generated from (TRM SEQ NO) and 'b' is generated from ØD

Figure 10. Terminal Block (No. 4)

Disp to End	
Disp to next entry	- entry 1 (control)
Name 1,2	six character name
Name 3,4	
Name 5,6	
Name option	2 character option (4040-4047)
Spare 1=0	
Spare 2=0	
00 protection	(c.f. block 2)
User's ID	creator's ID, not current user
→Disp to next entry	- entry 2 (parameter model list)
Y	
X	
L ₁	standard text entry
Text ₁	(c.f. block 1)
L ₂	
Text ₂	
⋮	
→Disp to next entry	- entry 3 (type name list)
Y = 1344	
X = 0	
L ₁	standard text entry
Text ₁	(c.f. block 1)
L ₂	
Text ₂	
⋮	
→Disp to next entry	- entry 4 (terminal list)
AO (TYPE NAME NO)	A=1, type not assigned; =0, assigned.
Y ₁	(TYPE NAME NO) is index into entry 3
X ₁	of this block
AO (TYPE NAME NO)	List corresponds in order to external
Y ₂	terminals in structure definition
X ₂	
⋮	
→Disp to next entry	- entry 5 (lines)
dY	
dX	
PT ₁ Y	standard line entry
PT ₁ X	
PT ₂ Y	(c.f. block 0)
PT ₂ X	
⋮	



DC (Draw Area Center) = (1074, 714)

MC (Mnemonic Center) = (1000, 1000)

Draw Area = 640 by 904 points (4.5" by 7.5")

Characters = 8 by 12 points set in the lower left
corner of 12 by 20 point areas

The number to the left of each area name designates that area
(e.g., the Draw Area is screen segment 4)

Figure 13. Display Screen Composition Detail

<u>number</u>	<u>name</u>	<u>function</u>		
0	CALR	Logon/program menu (SYSTEM)	} (APPLICATIONS)	
1	GSAM	Simulation and Modeling		
2	LISD	Library Manipulation		
3	ARCH	Architectural Design Aid		
4	TEXT	Document Preparation		
5	(etc.)			
6				
7				
10				
11				
12				
13				
14				
15				
16				
17				
20				
21				
22				
23				
24				
<hr/>				
25	GND1	generalized drawing package		}
26	GND2			
27	GND3			
<hr/>				
30	GUT1	(mini text editor)	} Utility Functions	
31	GUT2	(regen, file, send, menu)		
32	GUT3	(REACT)		
33	DEX1	Logon directory extensions	} (SYSTEM)	
34	DEX2			
35	DEX3			
36	SPIR	(Spare)		
37	PSIR	IR initializer/blk sorter		

Figure 14. Program Segment Allocation

START

60

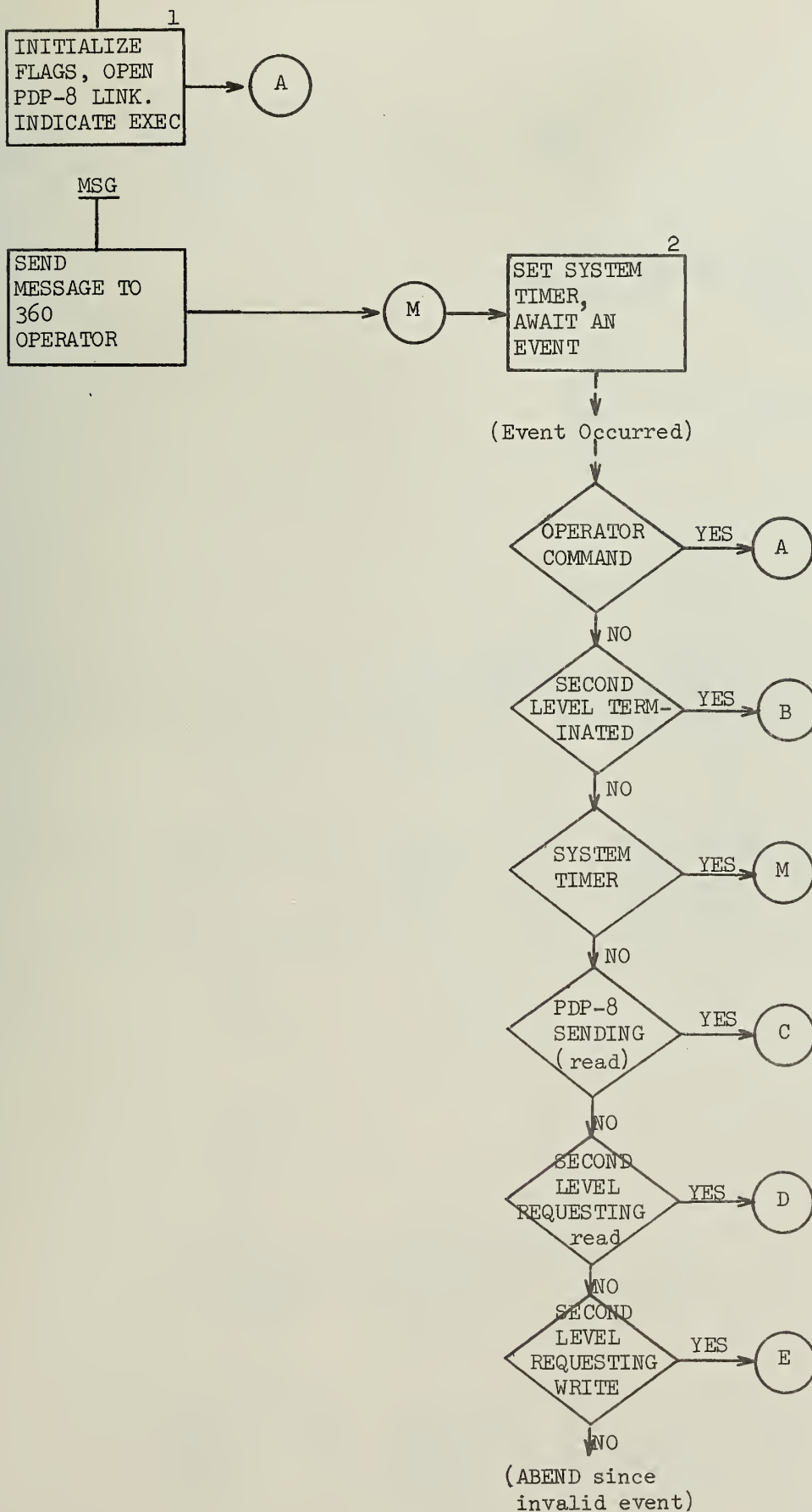


Figure 15. G80PERAT Flow of Control (Part 1 of 2)

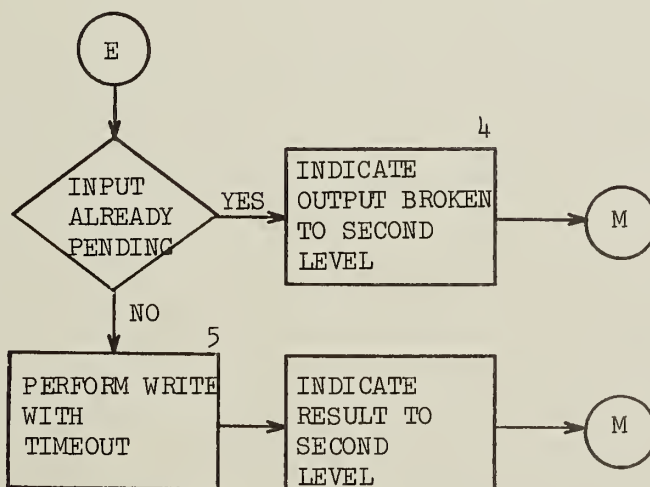
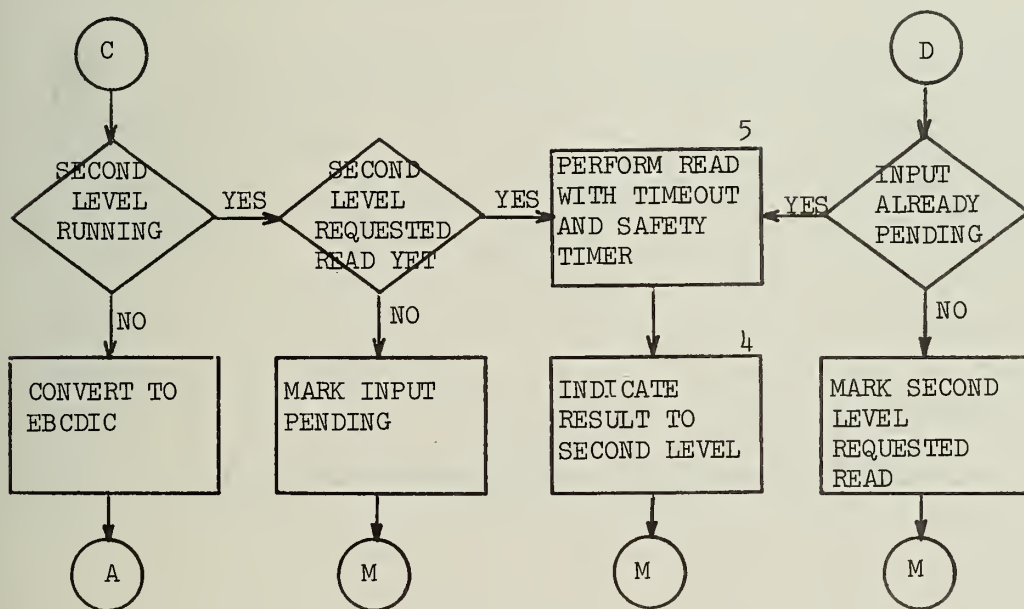
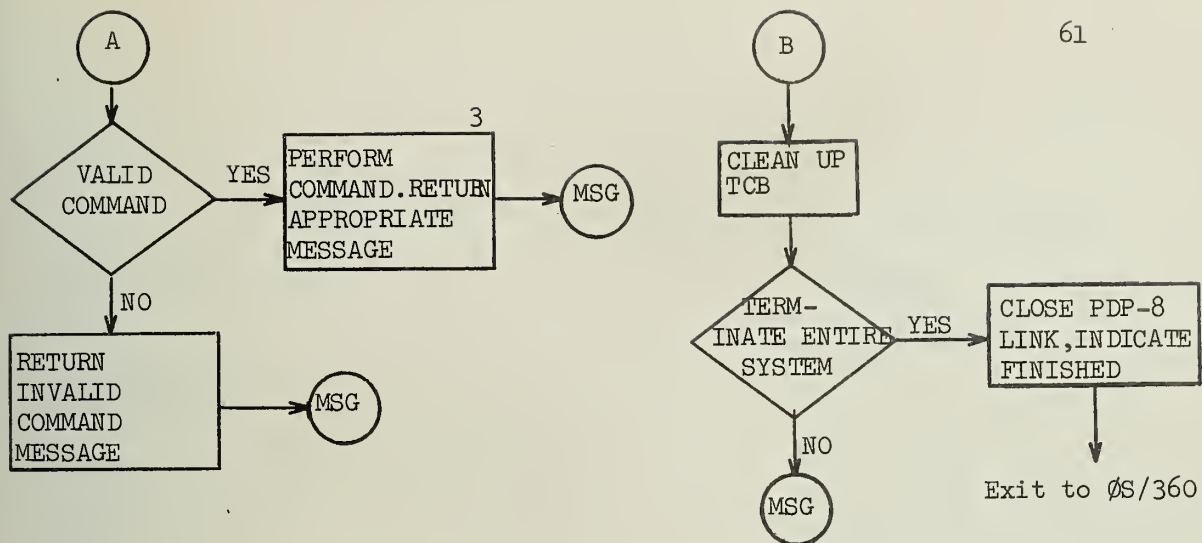


Figure 15. (Part 2 of 2)

After G8ØPERAT ATTACH of subtask, but before G8ENTRY call:

<u>Parmlist for subtask</u>	<u>Subtask</u>
+0 0	(Unknown)
+4 A(RDRQECB)	
+8 A(STRING)	

G8ØPERAT

RDRQECB = 0
WTRQECB = 0
(e.g., WTRQECB is at RDRQECB+4)

After G8ENTRY call, but before any subtask 2701 requests:

<u>Parmlist for subtask</u>	<u>Subtask (within G8ENTRY)</u>
+0 A(SUBTECB)	+0 SUBTECB = 0 (for G8ØPERAT)
+4 A(RDRQECB)	+4 A(RDRQECB) HALT notice)
+8 A(STRING)	

G8ØPERAT

RDRQECB = 0
WTRQECB = 0

Figure 16. G8ØPERAT/Subtask Linkage (Part 1 of 3)

After subtask request for 2701 read (G8READ):

<u>Parmlist for subtask</u>	<u>Subtask</u>
(same as above)	(same as above)
<u>G8OPERAT</u>	<u>Record Request Block 1</u>
RDRQECB = A(INECB)	+0 INECB = 0
WTRQECB = 0	+4 A(INBUF)
	+8 (reserved)
	+10 Byte length of INBUF
(subtask read request pending)	INBUF (subtask supplied area)

After 2701 read complete:

<u>Parmlist for subtask</u>	<u>Subtask</u>
(same as above)	(same as above)
<u>G8OPERAT</u>	<u>Record Request Block 1</u>
RDRQECB = 0	+0 INECB = return code
WTRQECB = 0	+4
	.. (same as above)
	+10
(subtask read request cleared)	<u>INBUF*</u>
	+0 console number (00N0)
	+2 record type (XXXX)
	+4 record len (PDP-8 words -1)
	+6
	.. Data
	+n

* All input in the unpacked PDP-8 format
(e.g., 00XXXXXX00XXXXXX = one 12-bit PDP-8 word)

Figure 16. (Part 2 of 3)

After subtask request for 2701 write (G8WRITE):

Parmlist for subtask

(same as above)

G8OPERAT

RDRQECB = 0
WTRQECB = A(ØUTECEB)

(subtask write request pending)

Subtask

(same as above)

Record Request Block 2

+0 ØUTECEB = 0
+4 A(ØUTBUF)
+8 (reserved)
+10 Byte length of ØUTBUF

ØUTBUF*

+0 console number (ØØNØ)
+2 record type (XXXX)
+4 record len (PDP-8 words -1)
+6
.. Data
+n

* All output in unpacked PDP-8 format.

After 2701 write complete:

Parmlist for subtask

(same as above)

G8OPERAT

RDRQECB = 0
WTRQECB = 0

(subtask write request cleared)

Subtask

(same as above)

Record Request Block 2

+0 ØUTECEB = return code
+4
.. (same as above)
+10

ØUTBUF

(same as above)

Figure 16. (Part 3 of 3)

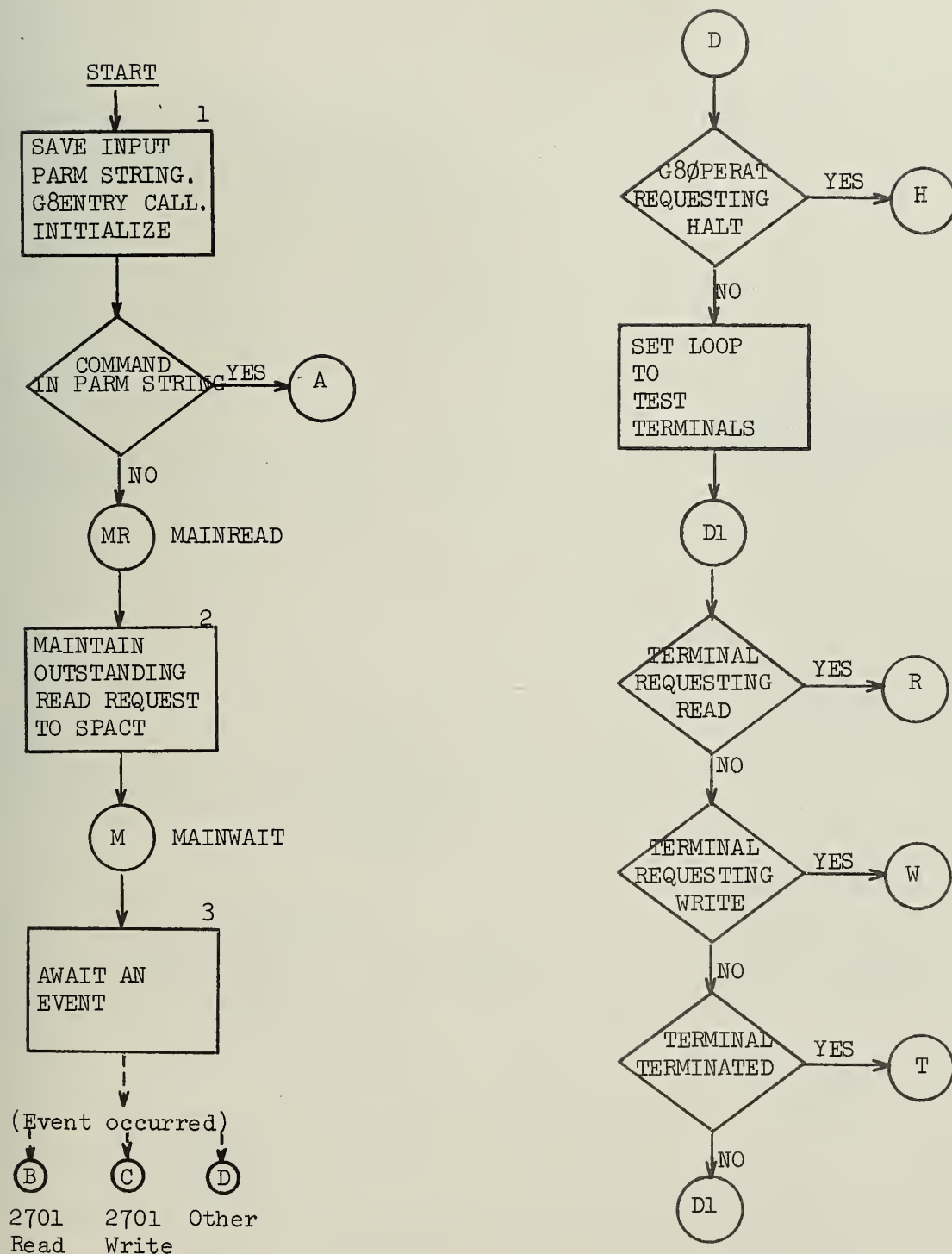


Figure 17. SPACT Flow of Control (Part 1 of 7)

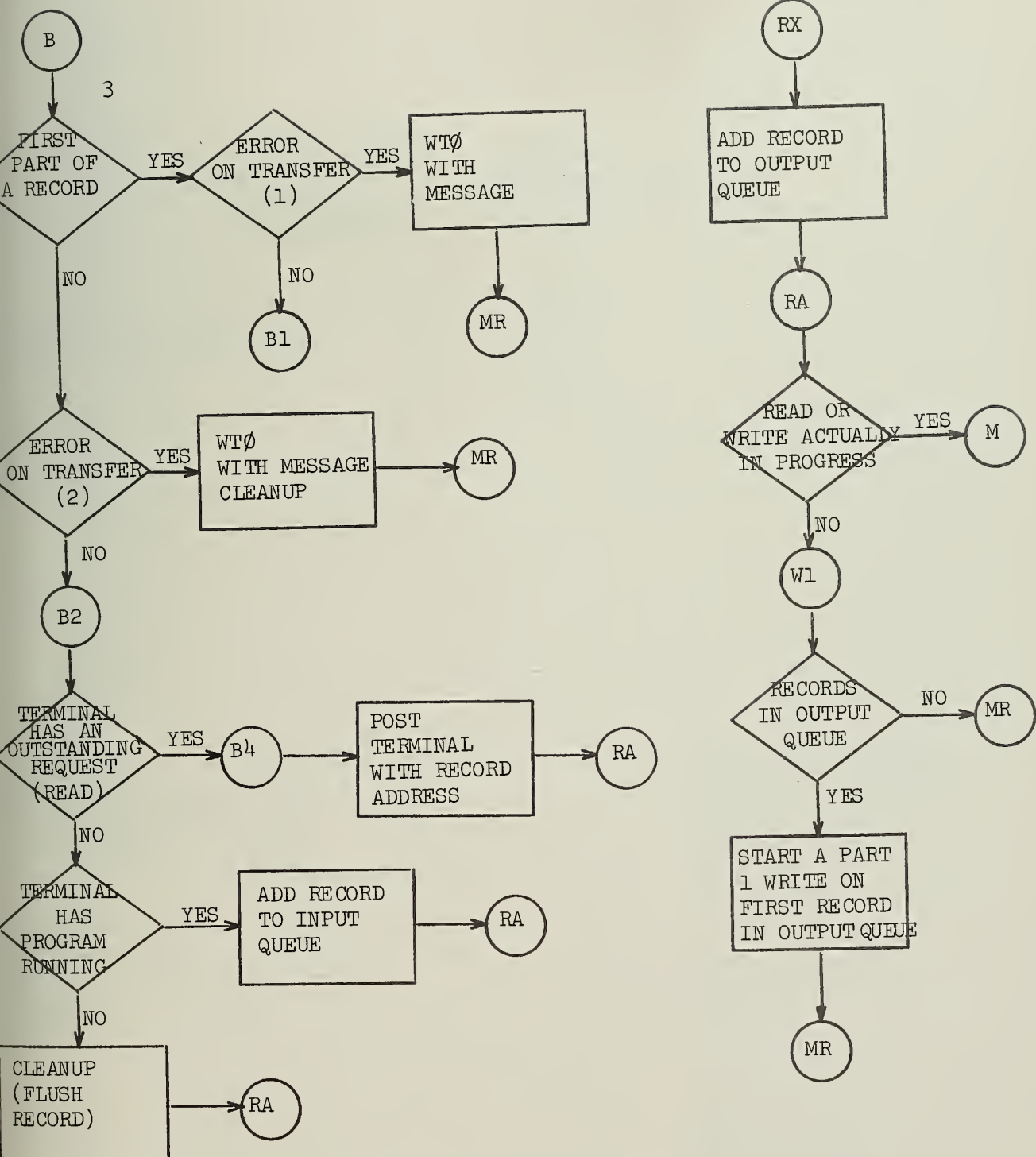


Figure 17. (Part 2 of 7)

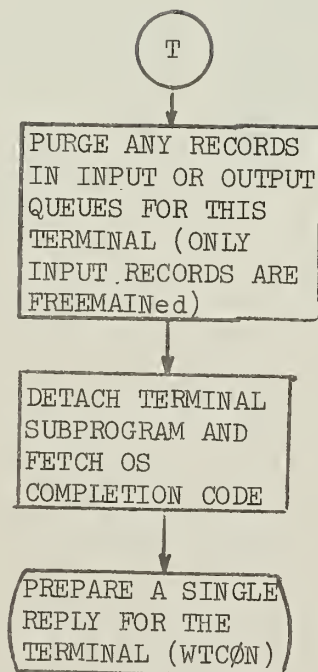
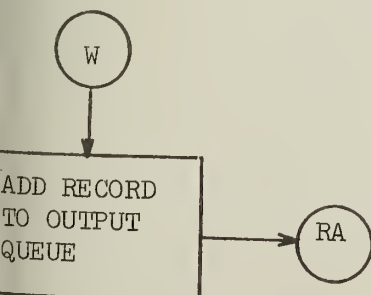
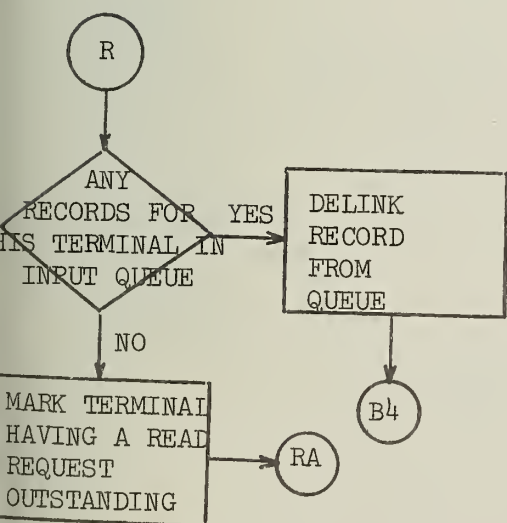
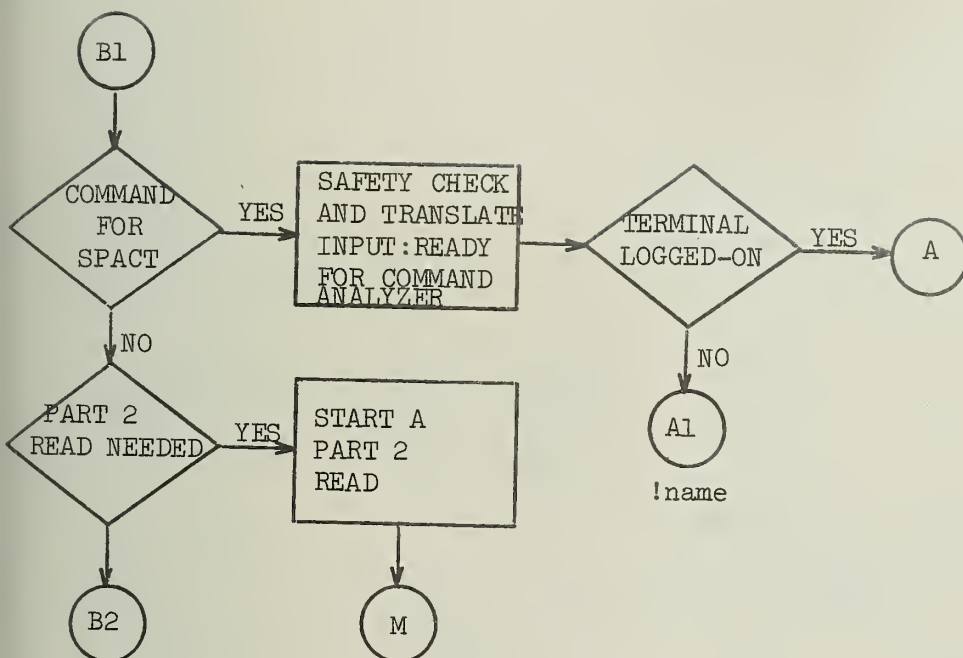


Figure 17. (Part 3 of 7)

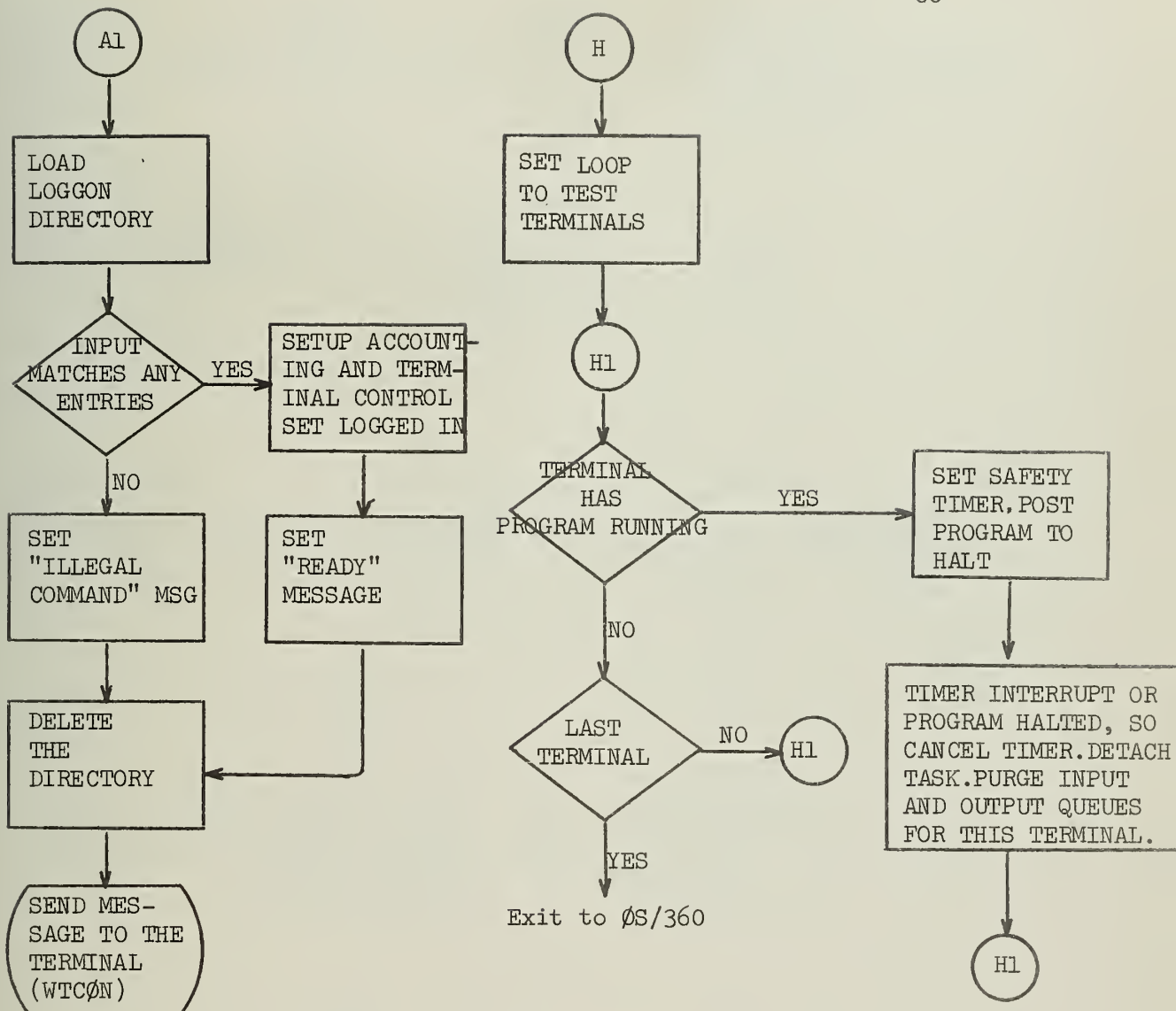


Figure 17. (Part 4 of 7)

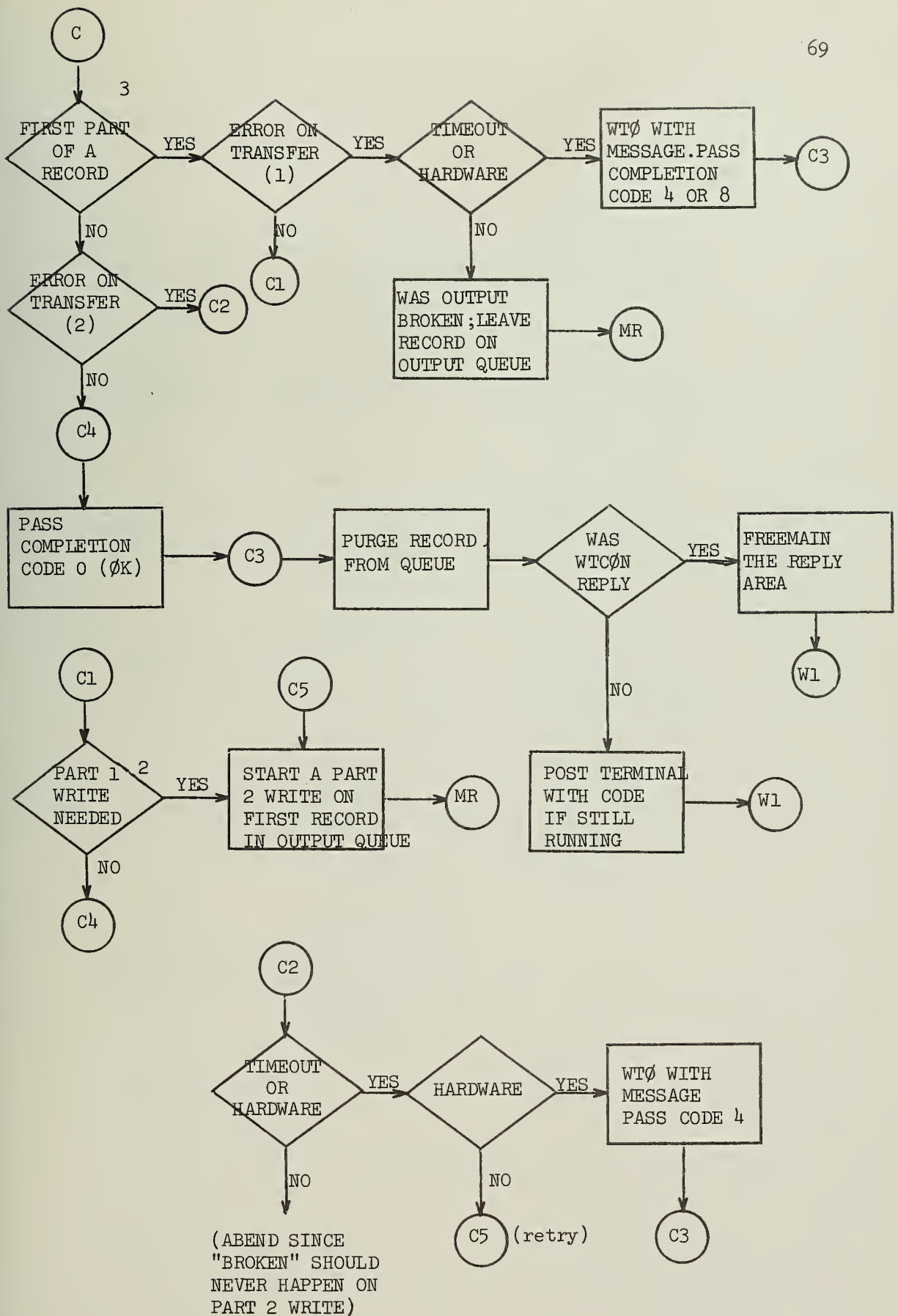


Figure 17. (Part 5 of 7)

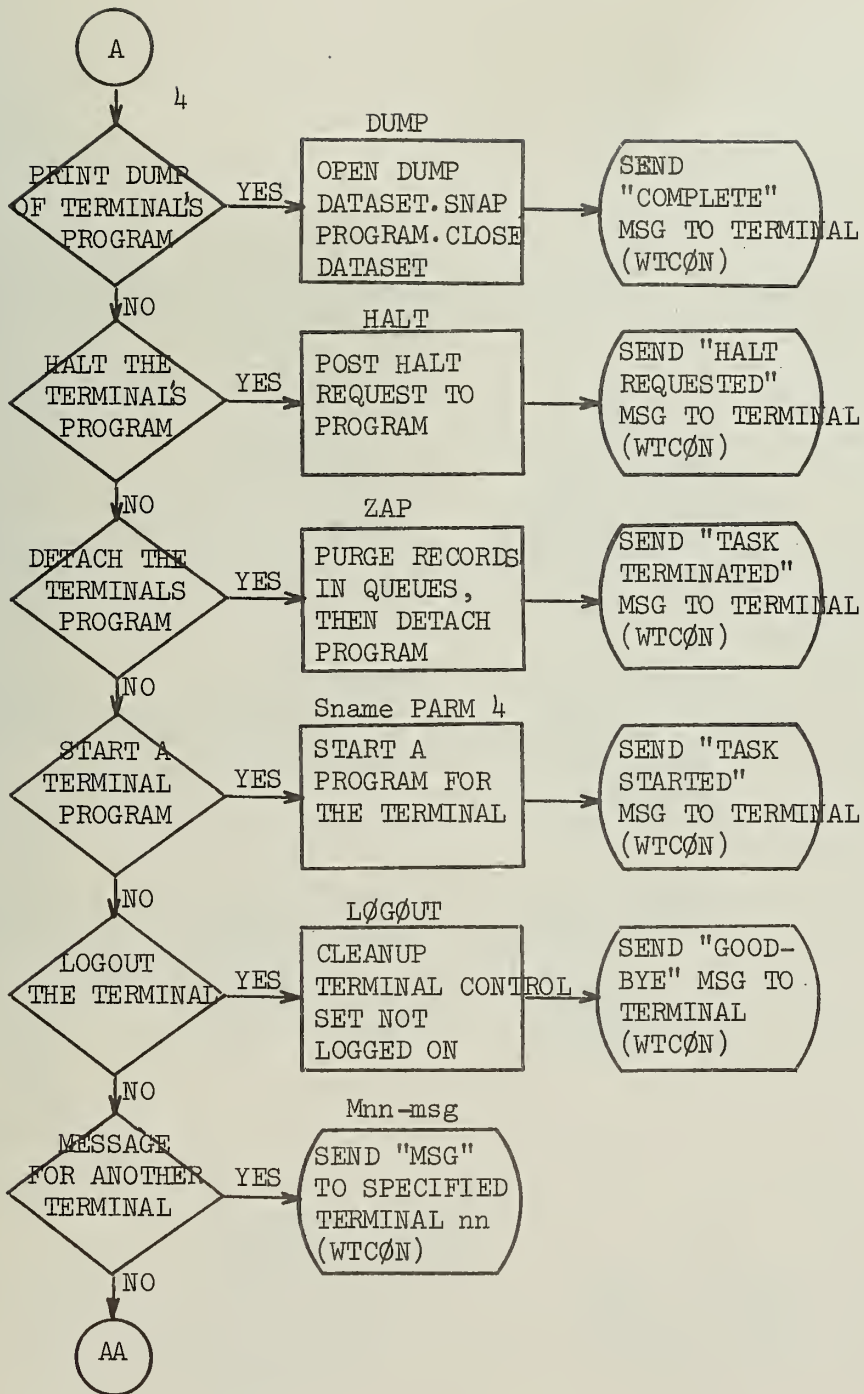


Figure 17. (Part 6 of 7)

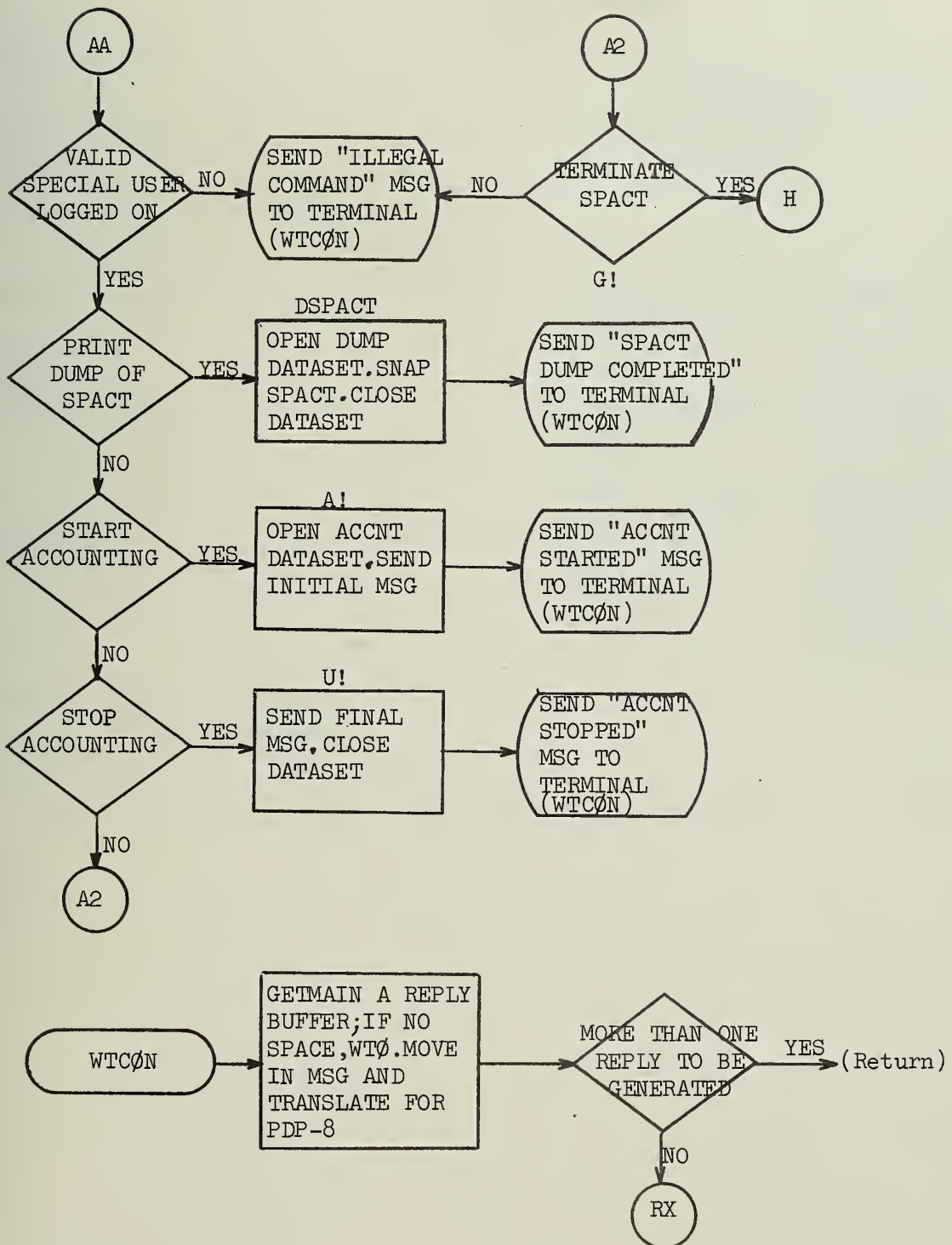


Figure 17. (Part 7 of 7)

<u>SPACT</u>	<u>Subtask</u>
<u>Parmlist for subtask</u>	(user must:
+0 (available)	1. save parmlist address
+4 A(user work area)	2. set XFILE001 into V-con
+8 A(XFILE001)	3. call S8ENTRY
+12 A(log user)	before anything else)
+16 A(parm string)	
 <u>Parmstring</u>	 <u>User Work Area (14 words)</u>
24 characters:	+0 0 SPACT ECB for
parm field	HALT request
(from START command)	+4 A(2701ØPRQ) 2701 request link
	+8 0 2701 Read done ECB
	+12 0 2701 Write done ECB
<u>Loguser</u>	+16 (+8,+12,+0) S8WAIT ECBLIST
8 characters:	+28 (+8,+0) S8READ ECBLIST
user logon name	+36 (+12,+0) S8WRITE ECBLIST
	+44 0 A(ØTHER)
	+48 0 Register 2 save wrd
	+52 0 Addressability
	save word
 <u>Trmcntrl</u>	
+0 2701ØPRQ: READ request ECB	
+4 WRITE request ECB	
+8 A(User work area) +8 ØPdone ECB's	

Figure 18. SPACT/User-Program Linkage After S8ENTRY

Trmcntrl

- +0 Read request ECB: PØSTed by subtask when S8READ issued.
- +4 Write request ECB: PØSTed by subtask when S8WRITE issued.
- +8 A(Subtask Op-completed ECB's): set by S8ENTRY. ECB's PØSTed by
SPACT as appropriate.
- +12 Subtask TCB: Used for DETACH and DUMP; if zero, no subtask running
(Saved after ATTACH)
- +16 Subtask End ECB: PØSTed by ØS when subtask terminates (this ECB
specified during ATTACH).
- +20 A(ACCNT): A(Accounting Area) for this terminal.

Accnt

- +0 User logon string
- +8 Status word one
- +12 Status word two } record counts, funds, etc.
- +16 Control byte (X'80' = logged on)
- +17 Data logged on
- +20 Time logged on

Figure 19. SPACT Terminal Control Area Detail

LIST OF REFERENCES

- [1] Michel, M. J. GRASS: System Overview, Department of Computer Science Report No. 465, University of Illinois, Urbana, Illinois 61801, July 1971.
- [2] Gear, C. W. An Interactive Graphic Modeling System, Department of Computer Science Report No. 318, University of Illinois, Urbana, Illinois 61801, April 1969.
- [3] Gear, C. W. Generalized Simulation and Modeling System, Department of Computer Science File No. 785, University of Illinois, Urbana, Illinois 61801, December 1968.
- [4] Michel, M. J. GRAPHICAL REMOTE-ACCESS SIMULATION SYSTEM (GRASS) The Communication and Monitor Components (GASP/GLASP), Department of Computer Science Report No. 346, University of Illinois, Urbana, Illinois 61801, August 1969.
- [5] Gear, C. W., et.al. The Simulation and Modeling System---A Snapshot View, Department of Computer Science File No. 824, University of Illinois, Urbana, Illinois 61801, February 1970.
- [6] Levin, H. Local Information Retrieval for the Simulation and Modeling System, Department of Computer Science Report No. 409, University of Illinois, Urbana, Illinois 61801, August 1970.
- [7] Michel, M. J. and Koch, J. GRASS: Terminal User's Guide, Department of Computer Science Report No. 467, University of Illinois, Urbana, Illinois 61801, August 1971.
- [8] Haskin, R., Nickolls, J., and Michel, M. J. GRASS: Remote Facilities Guide, Department of Computer Science Report No. 466, University of Illinois, Urbana, Illinois 61801, July 1971.
- [9] Michel, M. J. and Haskin, R. GRASS: Extended Remote Facilities Guide, Department of Computer Science File No. 867, University of Illinois, Urbana, Illinois 61801, August 1971.
- [10] Haskin, R. GRAPHICS 8: System Maintenance (Software), Department of Computer Science File No. 865, University of Illinois, Urbana, Illinois 61801, July 1971.
- [11] Nickolls, J. GRAPHICS 8: System Maintenance (Hardware), Department of Computer Science File No. 866, University of Illinois, Urbana, Illinois 61801, August 1971.
- [12] Carter, C. E., et.al. GRAPHICS 8: Graphics Hardware--1469 Gear, Department of Computer Science Report No. 371, University of Illinois, Urbana, Illinois 61801, December 1969.

- [13] Carter, C. E., et.al. DEC PDP-8 Interface to IBM 2701 PDA,
Department of Computer Science Report No. 372, University of
Illinois, Urbana, Illinois 61801, March 1970.
- [14] Lopeman, H. E. GRAPHICS 8: PDP-8 Interface Hardware--1469 Gear,
Department of Computer Science File No. 828, University of
Illinois, Urbana, Illinois 61801, March 1970.
- [15] Levin, H. and Lopeman, H. PDP-8/I to PDP-8 Interface, Department
of Computer Science File No. 843, University of Illinois,
Urbana, Illinois 61801, June 1970.
- [16] Hyde, C. PDP-8 Disk Interface, Department of Computer Science
File No. 861, University of Illinois, Urbana, Illinois 61801,
November 1970.
- [17] Programmed Buffered Display 338, Digital Equipment Corporation,
DEC-08-G61C-D, 1967.
- [18] Disk Monitor System Programmers Reference Manual, Digital Equipment
Corporation, DEC-D8-SDAB-D, 1968.
- [19] System 360, International Business Machines Corporation, 1966-
1971 series.
- [20] COMPU TEK User's Manual, Series 400/15 CRT Display System, Computek
Inc., Bulletin 400M, July 1969.
- [21] Quarterly Progress Reports, Department of Computer Science Section 3.3,
University of Illinois, Urbana, Illinois 61801, March 1970-
June 1971.

APPENDIX A

Buffering Control Program (ACID)

A program has been developed for the PDP-8/I to maintain the storage tube terminals that are attached to it. Basically, display files are read out of the PDP-8, buffered in the PDP-8/I, and then fed to the terminal. In the other direction, joystick (X,Y) and keyboard text-lines input by a user at a terminal are sent to the PDP-8. A chained buffering scheme with storage allocation/deallocation is used, so that all attached terminals can be sent display data simultaneously. In addition, the line that a user types is refresh displayed in write-through (non-storing) mode, enabling typed corrections to appear immediately and in place. Moreover, since the line is not stored in this "line buffer" area, the display need not be erased and regenerated for the user to type his next line in the buffer.

Communication between the two CPU's is accomplished with two software flag words in the PDP-8, one for Input (terminal to PDP-8), and one for Output (PDP-8 to terminal).

(TFLAG8) Input: 1. zero means no input yet.

2. set to nonzero by PDP-8/I means input ready to be processed by PDP-8.
3. set to zero by PDP-8 means processing of input now completed; ready for more; go to state 1).

(DFLAG8) Output: 1. zero means no output yet.

2. set to nonzero by PDP-8 means output ready to be processed by PDP-8/I.
3. set to zero by PDP-8/I means processing of output now completed; ready for more; go to state 1).

TFLAG8 is located at 2000 in bank 3 (GLBANK), while DFLAG8 is at 50 in bank 3. TFLAG8 is the first word of a 40 word buffer; DFLAG8 is the first of three words.

For the following discussions, bits in a word are numbered most significant (left) to least significant (right), 0 to 11.

Flag Designations

A. TFLAG8: bit 0 (input ready when on)
 bit 1-8 (interrupt character)
 0 = carriage return (text line)
 1-7 = joystick hit (X,Y)
 8-255 = function option (currently unspecified)
 bit 9-11 (console number)

Rest of Buffer:

<u>keyboard line</u>	<u>Joystick hit</u>
word 2 number of characters (max currently allowed = 72)	Y (10 bits)
word 3 (unused)	X (10 bits)
word 4-40 six bit packed ascii characters with last set to a "blank" if count (word 2) is odd	(unused)

B. DFLAG8: bit 0 (output ready when on)
 bit 1-5 (function)
 1 = enable console
 2 = input text line to console's
 text-line display buffer
 3 = process display file for
 console (e.g., display)
 4-31 (unspecified)

Display File

The display file input by the PDP-8/I is in the DEC 338 file format [17], except for the use of the "spare" mode 7. This mode is used to provide special display instructions to the PDP-8/I. Each bit (of bits 3-11) represents an option and the bits are "executed" right to left, so "microprogramming" of options is possible. (option is "on" if bit is a "1").

bit 11 - disable output of terminal to PDP-8
 bit 10 - erase screen
 bit 9 - add frame lines to display
 bit 8 - enable output of terminal to PDP-8
 bit 7 - end of display file segment (e.g., another
 file will be output soon for this terminal; do
 not release buffers)
 bit 6 - (same as bit 7 currently)
 bit 3, 4, 5 currently unspecified

Terminal Character Set

For the following, ↑ represents a control character and - means the indicated character is ignored. All numbers are octal.

<u>typed on keyboard</u>	<u>echoed in line buffer as</u>	<u>sent to PDP-8 as</u>
␣ (2)	move text-line cursor to (Mtlct) 1st line character	Not sent
␣ (5)	Mtlct last line character	Not sent
␣ (6)	Mtlct next right character	Not sent
␣BAC (10)	Mtlct next left character	Not sent
␣TAB (11)	Mtlct 6th right character	Not sent
␣K (13)	Kill character line	Not sent
CR (15)	Mtlct 1st line character, reset	Send line
F1 (21)	-	Joystick hit
F2 (22)	-	Joystick hit
F3 (23)	-	Joystick hit
F4 (24)	-	Joystick hit
F5 (34)	-	Joystick hit
F6 (35)	-	Joystick hit

(continued)

space (40)	space	40
! (41)	!	41
" (42)	"	42
# (43)	#	43
\$ (44)	\$	44
% (45)	%	45
& (46)	&	46
' (47)	'	47
((50)	(50
) (51))	51
* (52)	*	52
+ (53)	+	53
, (54)	,	54
- (55)	-	55
. (56)	.	56
/ (57)	/	57
0 (60)	0	60
...
9 (71)	9	71
: (72)	:	72
; (73)	;	73
< (74)	<	74
= (75)	=	75
> (76)	>	76
? (77)	?	77
A (101)	A	01

(continued)

...
z (132)	z	32
[(133)	(50
\ (134)	⌞	36
] (135))	51
___ (137)	___	35 but NOT sent; can only come FROM the PDP-8
a (141)	a	01
...
z (172)	z	32
{ (173)	(50
(174)		34
} (175))	51
⌞ (176)	⌞	36

Note that 0, 33, and 37 are never sent to the PDP-8 since these mean carriage-return, line-feed, and escape, respectively, when they appear in a display file. In addition, 35 (underscore) is never sent to the PDP-8. All possible keyboard characters not mentioned above explicitly are ignored by ACID.

APPENDIX B

System Generation and IPL

1. Local System Generation

Use a PDP-8 Disk Monitor System [18] to assemble and save the programs as shown below. PEND contains just a \$ to mark the end of PALD input. PSEQ contains a large number of equates; this facilitates symbolic references to all control areas and between separately assembled programs.

GLASP: GMN2,PEND

.SAVE GMN2!30000-35777;30203

IR: PSEQ,DISK,DIRC,SVIR,PSIR,PEND

.SAVE GLIR!26000-27777;26143 (moved to GLBANK during system IPL)

.SAVE PSIR!34000-35777;30150 (segment 37)

GUTS: UTEQ,GTS1,DSM1,DSM2,DSM3,DSM4,DSM5,IRRE,GTS2,2701,DPU1,PRIN,
DPU2,DSM6,PEND

.SAVE GUTS!10000-17777;10043

Segments: PSEQ,name,PEND

.SAVE name!34000-35777;30150 (segments 0-36)

ACID: ACID

.SAVE ACID:0-3777;200

A system load tape is then built from these modules using WLØD as described below. GLØAD and GEXIT for loading the system, and the library, are described in [7].

WLØD

- a. Mount the tape to be modified on Unit 4, write enabled.
- b. Start execution of WLØD from the DMS.
- c. Type the name of the contiguously saved program to be written on the tape.
- d. Type the octal number associating the program with an area on the tape. See following chart.
- e. Go to C) and repeat until all desired programs written; then type control C to exit to DMS. Unload tape.

<u>Data</u>	<u>WLØD No.</u>	<u>Blocks on Load Tape</u>	<u>Blocks on Disk*</u>
DECTape System**	-	0-377	-
Program Segments	0-37 ⁺	400-777	1140-1537
GLASP	40	1000-1024	-
GLIR	41	1025-1034	-
GUTS	42	1035-1074	-
ACID	43	1075-1114	-

<u>Data</u>	<u>Blocks on Library Tape</u>	<u>Blocks on Disk*</u>
Core copy of free block list (64 words)	0	--(from GLIR)
Control Pointers (6 words)	0	--(from GLIR)
Free block list	1-20	1540-1557
Directory blocks	21-120	1560-1657
Library blocks	121-2637	1660-4377

One other utility is available for tape modification. PEEPER allows octal alteration (in a manner similar to XOD or HELP) of arbitrary tape blocks.

PEEPER

- a. Start execution of PEEPER from the DECTape system (e.g., on a GRASS load tape).
- b. In response to "PGM#:", respond with
 1. a WLØD number (0-43) if a GRASS program modification is desired, or

* Consult [6].

+ Corresponds to program segment numbers.

** Includes: XØD, XDUP, XFILE, etc., as well as GLØAD, GEXIT, PEEPER

2. a DECTape unit number (D0-D7) if an arbitrary tape block is desired. In this latter case, an octal block number also must be specified when "BLK#:" is requested.
- c. The program (1) or tape block (2) will be read into core.
- d. Now proceed as with XØD to examine and change locations. Obviously, locations outside the range of the requested program or tape block (0-177) cannot be modified.
- e. Typing control-P or control-C will write out any changes made; the former will then cause PEEPER to restart at step b), while the latter causes return to the monitor.

2. Remote System Generation

The remote system makes use of three OS/360 data sets. A macro library, used only during the assembly of system or user programs, contains the following:

<u>P8-macros</u>	<u>S8-macros</u>	<u>XFile macros</u>
P8IØ	S8DATA	XAPPEND
P8ØPRS	S8ENTRY	XCLØSE
P8ØPRSX	S8READ	XDELETE
P8RCCW	S8SETUP	XDESTRØY
P8RETRY	S8WAIT	XINIT
P8WAIT	S8WRITE	XLINK
P8WCCW		XLIST
		XØPEN
		XREAD
		XREADDIR
		XREADIRS
		XREADN
		XREADS
		XREWIND
		XTERM
		XWRITE
<u>G8-macros</u>	<u>Utility macros</u>	
G8ENTRY	PALS	
G8READ	PBR	
G8WAIT	PLR	
G8WRITE	PLST	
	PSTR	

Two other data sets are active at execution. The first is the library data set used by XFILE. This contains the remote directory, space allocation bit map, and library blocks. The second is a load module library containing the system programs and all user programs that run under it. Modules included are:

<u>System</u>	<u>User</u>	
G8ØPERAT	LSØ	
SPACT	ITEM	} Simulation and Modeling Package
GIDS	GLØBALL1	
XFILE	GLOBAL2	
CØMMUNE	ELIM	
BATCH	CØMP1	
PØT	SPARSE	
	...	

Some other modules in this data set are used during linkediting:

CALLER
S8SETUP
MY#FIØCS

Note that all software runs in HIARCHY=1 and is system independent for release 18 and above of ØS/360.

Sample JCB

1. Create top monitor level

```
// EXEC ASM,MACFILE='MACRØS'
//          (G8ØPERAT Source)
// EXEC  LKED,LIBFILE='LØADMØDS',PARM='LIST,      X
//          XREF,LET,HIAR'
//          HIARCHY 1,G8ØPERAT
//          NAME  G8ØPERAT(R)
```


2. Create a typical lower level or user module

```
// EXEC ASM...
      (source)
// EXEC IKED,PARM='LIST,XREF,LET,RENT'
      NAME XX(R)
```

(The above module is loaded by a higher level module into the proper storage; hence, HIARCHY parameters need not be used).

3. Create a typical user module containing CALLER and special FORTRAN I/O

```
// EXEC ASM or FØRT...
      (source)
// EXEC IKED,PARM=...
      ENTRY CALLER
      INCLUDE SYSLIB(MY#FIØCS)
      NAME XX(R)
```

(Consult [8]).

4. Run the monitor system

```
//JØBLIB DD DSN=LØADMØDS,DISP=SHR
//GRASS EXEC PGM=G8ØPERAT,
//      REGION=(12K,250K),
//      PARM='S SPACT'
//SYSUDUMP DD SYSØUT=A
//SYSPRINT DD SYSØUT=A
//PDP8DD DD UNIT=021'
//DISK DD DSN=GRAPHLIB,DISP=ØLD
//FT05F001 DD DUMMY
//FT06F001 DD SYSØUT=A
//FT07F001 DD DUMMY
```


5. Run the PØT plotting package

```
//JØBLIB DD DSN=LØADMØDS,DISP=SHR
//PØT EXEC CALCØMP,
//      REGION=(175K,30K),
//      PARM='EP=PØT,PI=30,TIME=6.00'
//GØ.SYSIMØD DD DSN=LØADMØDS,DISP=SHR
//GØ.SYSPRINT DD SYSØUT=A
//GØ.SYSUDUMP DD SYSØUT=A
//GØ.DISK DD DSN=GRAPHLIB,DISP=ØLD
//GØ.FTO5FO01 DD DUMMY
//GØ.SYSIN DD *
          (input cards)
```

6. Run the BATCH test module

(Consult [9]).

7. Save the remote library on tape

```
//SAVE EXEC PGM=IEBGENER
//SYSPRINT DD SYSØUT=A
//SYSUT1 DD DSN=GRAPHLIB,DISP=SHR,
// DCB=(BLKSIZE=793,LRECL=793,RECFM=F)
//SYSUT2 DD DSN=SAVØNTAP,UNIT=TAPE,LABEL=i,
// DISP=(NEW,KEEP),VOL=SER=tapeid,
// DCB=(BLKSIZE=793,LRECL=793,RECFM=F)
```

8. Reload the remote library from tape

```
//RELØAD EXEC PGM=IEBGENER
//SYSPRINT DD SYSØUT=A
//SYSUT1 DD DSN=SAVØNTAP,UNIT=TAPE,LABEL=i,
// DISP=ØLD,VOL=SER=tapeid,
// DCB=(BLKSIZE=793,LRECL=793,RECFM=F)
//SYSUT2 DD DSN=GRAPHLIB,DISP=ØLD,
// DCB=(BLKSIZE=793,LRECL=793,RECFM=F)
```


APPENDIX C

GLASP Routines--Detail

INTHND

Entered via location 2, bank 0, with standard DEC interrupt procedure. First tests disk done and immediately goes to IR entry point BSRTRN (6000) if found. Next tests DPU software flags and goes to DPUINR if (1) DPU input not currently being processed (DPUINP=0) and (2) new DPU input is in buffer (DPUINF NOT 0). Clock flag is tested next; if on, all timer queue elements are incremented with INTHN2 (CLKQR). If INTHN2 finds any terminals to be interrupted, CLKQR3 is used to set 2010 pending for each such terminal. Note that the interrupt handler is NOT exited during this processing (as opposed to calls to BSRTRN and DPUINR). INTHN2 returns to the mainstream of INTHND processing at INTHN3, but after ELAPC is called to update the time display area for the system message line. INTHN3 is the final phase of interrupt processing; two tables are used to test all remaining device flags. The first table contains the IOT, while the second contains the appropriate address in bank 1 (UTBANK) for device handling if the associated IOT "skips" (flag on). Note that undesirable device flags can be ignored simply by using a "clear" IOT instead of a "skip" IOT in the first table. The last entry in the first table is a jump to the WAIT routine; hence, if only the clock or an undesired device flag caused the interrupt, GLASP returns to the wait state.

FR2701

Entered from INTHND if "2701 ready to send to PDP-8" was found. Uses preset data to begin the read operation, then goes to WAIT. When the operation is complete, FD2701 is entered from INTHND. This routine tests for and begins a retry if necessary (data error, etc.). If the record was correct, the routine goes to FN2701 to check if it was the first or second part of a record. The I2701P flag is set accordingly. If the record was a second part, FE2701 is entered to reset the system for another first part; GLASP is then called via OPDONE (e.g., 2701 completion service is finished). If the record is a first part, IOT information is preset for use by FR2701 in case a second part is expected. Then the record type is tested for a monitor-monitor internal command. If such is found, the appropriate

internal routine is called; if not found, the record is for a terminal, so FC2701 is called. This routine calls F2701W in GLBANK. F2701W tests the status of the terminal with TØLAC (see below). If the terminal is offline, FC2701 is told to flush the record internally and return to WAIT. If the terminal is online but inactive, FC2701 flushes the record, but causes the subsequent execution of segment 0 (CALLER) for the terminal (via SYSCTL). If the terminal is active, 2001 is set pending for the terminal and lockout checking is performed. Finally, F2701W goes to WAIT.

The only internal monitor-monitor command currently in use is indicated by type=7770. This means the remote monitor is up; an "H" is printed on the operator's TTY printer and an internal flag is cleared (see ØPKB).

ØPKB

Entered from INTHND if a character on operator's TTY keyboard is sensed. All commands are currently one character long and execute immediately. The input character plus a carriage-return, line feed sequence are sent to the TTY printer. If the command is unrecognized, a "?" is printed. Current commands are:

- ! = ignore segment 2701 output operations (e.g., an internal flag (Q2701P) is set so the whole operation proceeds as normal except that no transmission takes place). Used when the remote monitor is not up.
- " = the internal flag is cleared so 2701 output operations will indeed transmit data. Used when GLASP is restarted and remote monitor is already up.

Operator device input to a terminal is not currently supported.

DPUINR

Entered from INTHND if DPU input ready to be processed. Calls PICSEG to test input data type; for keyboard lines PICSEG tests the terminal's filter word (+36 console vector), while for joystick hits PICSEG first determines the screen segment hit and then tests the appropriate

filter word. In either case, an accept/not-accept code is returned to DPUINR. TØLAC is then called to test the terminal's status. If the terminal is offline, the input will be flushed internally; control is returned to WAIT. If the terminal is online but inactive, the input is flushed but segment 0 (CALLER) is started on the terminal via a call to SYSCTL. If the terminal is active and the code from PICSEG was "accept," 2004 is set pending; control returns to WAIT.

CLKQR

Entered during INTHND processing of clock interrupts to increment timer queue elements. If a timed interval reaches zero for an active terminal that has "accept timer input" set, CLKQR3 is used to set 2010 pending for that terminal. After all terminals are checked, control returns to INTHN3.

RØUTX

Entered via JMS from segments or system to begin a service for the current terminal. The desired service group is indicated by the number in the ACC at entry. Necessary parameters for the service are assumed already in SVPRM1-8 in GLBANK; these are saved in the console vector. The caller's restart address and bank are also saved in the console vector. Availability of the desired service is then tested via RØUTXR. If the service is not free, RØUTX stores the service group number in the console vector (thus indicating that a request is pending) and goes to WAIT. If the service is free, RØUTXR copies the parameters to SVPRM1-8 in UTBANK, sets the service request word in the console vector to -1 (terminal busy, service being processed), sets the service busy (with the address of the caller's console vector), and jumps to the RØUTX entry point of the routine in UTBANK. The console vector address is passed to the UTBANK routine in the ACC.

TSTQUE

Entered from WAIT to test all "pending" conditions (e.g. device input and service requests). Each console vector is tested; if the busy flag is on (request word = -1), no action is taken. If the flag is > 0 (e.g., a service request is pending), RØUTXR is called to try to start the request. A successful start by RØUTXR obviously terminates TSTQUE processing. Otherwise, testing the next terminal follows the RØUTXR call. If the flag is 0 (e.g., terminal free (not busy)), the device input pending bits are tested. If a bit is found on, the corresponding device routine address is checked; zero means ignore, nonzero means accept. TSTQUE has a list of "ignore" entry point addresses, one for each device. The appropriate routine is called, and the input is internally flushed. In the case of "accept," the nonzero flag is the entry address; the current terminal segment is started via XCTL (TSTQUE processing is terminated). If no action is needed or none can be initiated by TSTQUE, control returns to WAIT.

WAIT

Entered whenever processing must be suspended pending completion or occurrence of an operation or event. A group of priority flags are tested first; this enables certain critical internal processes to ensure serial use of particular resources. If a priority flag is set (not zero), the appropriate resource busy flag is tested. A zero (not busy) resource flag causes control to be passed to the routine address specified by the priority flag. The called routine is responsible for clearing and setting the priority flag, as necessary. WAIT processing is at an end. If no action is taken during priority flag testing, TSTQUE is then called to schedule pending requests or input if possible. If TSTQUE returns to WAIT, the interrupt is turned on (IØN) and the program loops until an interrupt occurs (e.g., a device or the clock). This is the one and only idle state for the entire local system.

ØPDØNE

Entered from a completing service routine or from XCTL (see below). As soon as possible, the segment for the indicated terminal is restarted at the address (and bank) stored in the console vector. The ACC tells ØPDØNE which service is completing (0 for XCTL); the terminal to be restarted is indicated by the contents of the service's busy flag. This word is used to reset CURCØN (current console), and the service flag is set to zero (free). TSTPCR is then called to check if the segment needed for this terminal is the one currently in core. If it is, the SVPRM save area in the console vector is copied into SVPRML-8 in GLBANK; then the return information is extracted. Control is passed to the restart location; ØPDØNE processing terminates. If the required segment is not in core, the console vector address for this terminal is placed in a stack, and a priority call to WAIT is made (GETPl). ØPDØNE processing terminates temporarily.

As soon as the disk busy flag is clear, the priority flag scan in WAIT will transfer control back to ØPDØNE. The last entry on the console address stack is extracted, and, if the stack is now empty, the priority request is cleared. ØPDØNE now calls FETPA to load the segment needed by the indicated terminal from the disk. When the segment is in core, ØPDØNE processing proceeds as above. Note that the disk busy flag is made "busy" when FETPA is called; hence, WAIT will never make a priority return to ØPDØNE while a segment is being fetched.

RELEASE

Entered from a segment when current processing of input has been completed. The terminal (e.g., segment) is now ready (and waiting) for more input. RELEASE sets the request word to zero (free), and then performs a safety "unfix" via MNUNF for all standard, nonstandard, and work blocks associated with the terminal. Control is then passed to WAIT.

XCTL

Entered to begin execution of a different segment for the current console. The call is:

```
TAD X
JMS I XXCTL
ADDR
ID
```

ADDR gives the new entry point; ID (0-37) gives the new segment number. If $X < 0$, the SVPRM1-8 area in GLBANK is saved in the console vector before the operation commences; otherwise, the GLBANK copy is lost (before execution of the new segment begins, the console vector SVPRM1-8 area is copied to the GLBANK area). XCTL simply resets the locations used by RØUTX and calls ØPDØNE with ACC = 0. If ID = -1, the segment is not changed; the current segment is entered at the new location. One routine mentioned previously, SYSCTL, is merely a call to XCTL that specifies CALLER as the segment and 4000 as the entry address.

GLMSG

Entered to take an indicative text-line from the 2701 and display it on the appropriate terminal. The call is a JMP to GLMSG with SVPRM3 = Y and SVPRM4 = X giving the screen position of the line. GLMSG returns to RELESE when finished, not to the segment. The appropriate call is made to RØUTX (service 4) to complete 2701 processing. If the input is not of type = 0, the data is flushed, and a message is sent to the terminal. If the input is type = 0, the input appears on the terminal. PCWØRK is used to buffer the input; the 2701 is cleared; and then the DPU is enqueued. When DPU processing is complete, the word at PCWØRK is set = 0, and RELESE is called. Since some segments (for example the text editor in GUT1) use PCWØRK for user area message lines, setting PCWØRK = 0 will prevent a DPU error when such a segment is re-entered following GLMSG processing (c.f. the text editor and GREGP1). Note that a segment can enter this routine at a point GLMSG for displaying a line already in PCWØRK.

DPUDIN

Entered to transfer control to the proper routine after DPU input marked pending. Segments desiring to accept DPU input should always use DPUDIN as the device entry address (+22 in the console vector). TSTQUE will transfer control to DPUDIN as appropriate; DPUDIN uses information saved by PICSEG to transfer control to the proper screen segment or keyboard line routine. DPUDIN always moves words at DPUINF+1 and DPUINF+2 into +112 and +113 of the PCB. For joystick DPU input, these are the Y and X hit coordinates.

GREGEN

Entered to automatically redisplay the standard blocks of the current picture. The routine assumes that an erase, the frame, and any desired frame text have already been sent by the segment. The line, text, terminal, and then instance blocks are added to the screen via calls to RØUTX, the DPU, and GUT2 (RGINST). Control is returned to the segment.

GREGP1

Entered to add a preformed display file (such as a text message or the mnemonic "box" in GNDRW) to display file being constructed in the DPU output buffer. The call is:

```
TAD X
JMS GREGP1
FILEH
...
FILEH, LEN
(Rest of file)
...
```

The TAD X is used ONLY if the file (FILEH) is not in GLBANK; in this case, X should = otherbank - GLBANK. The file is considered as an entity and is added directly onto the current file with DPUG. When the DPU buffer is complete, the segment should call:

```
TAD X
JMS GREGP2
```


where X is zero or any desired initial 700X display command. A DPUC call will be set up; the segment should follow this call with:

```
CIF CDF!UTBANK
JMS I XDPUDP
```

to send the display.

SVPUT

Pass a parameter to an SVPRM location in UTBANK:

```
TAD PARM
JMS SVPUT
SVPRMn           /n = 1-8
```

FREDPU

Clear the DPU flags (DPUINP and DPUINF) and set first eight DPU input buffer words to blanks (4040's). This allows the DPU to asynchronously store more input into the buffer; this call MUST be made as soon as the executing segment finishes using the data in the buffer. (JMS FREDPU).

RETIME

Reset the timer queue element for ten minutes. (JMS RETIME).

FREDTM

Calls FREDPU, then RETIME. (JMS FREDTM).

RLV1

Use the data stored at console vector location +76 to form a parameter list and call XCTL (ACC = 0). Some segments store return data in locations +72 (level 3), +74 (level 2), and +76 (level 1) of the console vector before invoking a segment subfunction; the subsegment uses this to return via RLV1, RLV2, or RLV3 to the calling segment. (JMP RLV_n).

MØVEDT

Move data between any two locations. At entry, the PDP-8 data field register specifies the bank of the calling program. Data is moved starting at the specified address and continuing until the word count reaches zero. The call is:

```
CIF!GLBANK
JMS I (MØVEDT
From addr
To addr
From bank (00F0)
To bank (00T0)
Count (> 0)
```

SADCV

Add the specified word in the current console vector to the ACC:

```
JMS SADCV
disp (0-77)
```

SSTCV

Store the ACC at the specified word in the current console vector:

```
JMS SSTCV
disp (0-77)
```

SADPC

Add the specified word in the current PCB to the ACC:

```
JMS SADPC
disp (0-177)
```

SSTPC

Store the ACC at the specified word in the current PCB:

```
JMS SSTPC
disp (0-177)
```


STØP

Reset the current console options area (+14...+37):

```
TAD  X
JMS  STØP
Addr
```

Use TAD X only to specify that less than 2^4_8 words are to be reset starting at +14 (e.g., $2^4 + X = \text{number reset}$). Addr is the location of the new contents (in GLBANK only!).

STWK

Reset the current console work are (+50...+77):

```
TAD  X
JMS  STWK
```

The use of TAD X is analogous to its use in STØP. The data moved in is assumed to be located at +150...+177 in GLBANK page 0.

LDWK

Reset +150...+177 in GLBANK page 0 from the current console vector work area (+50...+77):

```
JMS  LDWK
```

No alternate data length may be specified.

SVPRLD

Copy the current console vector SVPRM area (+40...+47) to the SVPRM area in page 0 (+140...+147) of the specified bank:

```
TAD  BNK
JMS  SVPRLD
```

BNK = 00n0, where n = 3(GLBANK) or 1(UTBANK).

SVPRSV

Copy the SVPRM area in page 0 (+140...+147) of the specified bank to the current console vector SVPRM area (+40...+47):

```
TAD  BNK
JMS  SVPRSV
```

BNK = 00n0, where n = 3(GLBANK) or 1(UTBANK).

TØLAC

Test current console status:

```
TAD  X
JMS  TØLAC
+0 (inactive return)
+1 (offline return)
+2 (active return)
```

The status of the terminal is indicated by the address of the return. If the terminal was inactive and the ACC was zero on entry to TØLAC, TØLAC will set the terminal to active before it returns at +0. If the ACC was not zero, TØLAC just returns at +0.

GLINE

Calculate a line number (1...n) for a given Y coordinate given the number of lines in the list and the Y coordinate of the top line. The input Y coordinate is assumed to be in +112 of the PCB (e.g., set by DPUDIN, etc.).

```
JMS  GLINE  (ACC MUST = 0)
List length
Top Y coord
```

If 0 is returned in the ACC, the input Y missed the list. This routine is obviously handy for menu picking.

APPENDIX D

GUTS Service Routines--Detail

1. DSMN Routines

1.1 List

Entries

MNDPC - Direct call entry
 MNNTRE - RØUTX call entry

RØUTX Calls

MNBTB - Block-to-block data transfer
 MNDEB - Delete a current picture block
 MNIBA - Initially allocate space and load new block
 MNILE - Insert data at end of last entry in current block
 MNINE - Insert data at end of current entry
 MNLØDR - Load a current picture block into core
 MNMNEM - Build a mnemonic block from the current picture
 MNMNIN - Integrate a mnemonic as an instance into the
 current picture
 MNNEN - Insert a new entry in the current block
 MNNNE - Insert a new entry at the end of the current block
 MNREAL - Reallocate (e.g., expand) a current picture block
 MNRED - Reduce the allocation of a current picture block
 MNRPL - Insert, delete, or replace data in the current block
 MNSTØR - Explicitly store a current picture block in an
 IR temporary

Direct Calls

MNBLØD - Load the word in the current block pointed to by BLKØRG
 MNBRS - Reset the specified block from its PCB entry as the
 current block (see MNBSV)
 MNBSET - Store the ACC in the word pointed to by BLKØRG
 MNBSV - Save the status of the current block in its
 PCB entry (see MNBRS)

MNDEN -	Delete the current entry in the current block
MNDTX -	Delete text-lines in the current entry
MNELØD -	Load the word in the current block pointed to by ENTØRG
MNELST -	Return the number of entries in the current block (calls MNDRVE)
MNENUM -	Return the number of the current entry (1...n)
MNESET -	Store the ACC in the word pointed to by ENTØRG
MNEXT -	Extract the specified text-line from the current entry
MNEXTC -	Extract arbitrary data at PØINTR in the current entry
MNFIX -	Mark the specified block as fixed
MNFPR -	Reset the PØINTR in the current entry
MNFPRA -	Reset the PØINTR to the beginning of the data +MNMØD in the current entry
MNFPRT -	Reset ENTØRG and PØINTR to the first entry in the current block and return the block's address (BLKØRG)
MNHTN -	Return the highest terminal sequence number +1 from the current block (assumed to be a terminal block) or return 0 if no terminals in the block yet
MNIBD -	Set the initial data for a block and mark the block initialized
MNINS -	Search for a Y, X match on an instance
MNLTH -	Move PØINTR to the specified K-th text-line (or the end) of the current entry
MNMØV -	Move the specified data into the area of PØINTR
MNMRK -	Mark the specified bits in the specified word of the current entry
MNMVP -	Move data (<u>≤</u> 5 words) from SVPRM ⁴ -8 into the area of PØINTR
MNNXTE -	Move ENTØRG and PØINTR to the next sequential entry in the current block and return an error if block end exceeded.

MNPERG -	Position ENTORG to the specified n-th entry in the current block
MNPLD -	Load the word at the current PØINTR location
MNPRG -	If the current block is marked in core and initialized, its core is released by a call to MNFRE
MNPSET -	Store the ACC at the current PØINTR location
MNTRL -	Search current (assumed terminal) block for terminals with the specified instance sequence number. Call MNSCAN
MNTR2 -	Same as MNTRL, but search is for specified terminal sequence number
MNTR3 -	Similar to MNTRL, but search is for specified Y, X and call is to MNYXF
MNTXS -	Calls MNYXF to search the current (assumed text) block for an entry that matches the specified Y, X
MNUNF -	Mark the specified block as unfixed
MNYNM -	Return the line number for the text-line in the current entry that corresponds to the input Y value
MN1EX -	Extract one word from the current entry at the specified location
MN2TB -	Move data from the system 2701 first-part record input buffer to the current block (essentially MNIBD); set 2701 input IØT data for the second-part of the record; and return the second-part input address

Internal Routines (some of these are NOT restricted to DSMN use)

CTIME -	Return least significant two decimal digits of an input binary number to GLBANK calling routine
DECML -	Call DECTYP to obtain four decimal digits of an input binary number; UTBANK call only.
DECTYP-	Convert input binary number to decimal

DSMNGØ - Calls DSMNG6 to access IR internally for writing, reading, and deleting block temporaries.
Called by MNSTØR, MNDEB, MNLØDR

DSMNG3 - Called from DSMNG6 to perform actual IR call.
Saves and restores pertinent terminal data; calls DSMNIR in GLASP to ensure serialized IR use (via GETDS WAIT priority call).

DSMNG4 - Same as DSMNGØ but called from MNLDST during MNLØDR processing (need more core).

DSMNG6 - Common routine to build IR parameter list for DSMNG3. Called from DSMNGØ and DSMNG4.

DTYPE - Called by DECTYP to pack four decimal digits into two words

MNADST - Extract a word in a table

MNADTB - Convert a block address to a block (MNGET "page") number

MNALC NOT a routine, but the MNGET page allocation list

MNBØST - Use the current BLKNUM to extract and set BLKØRG

MNBTAD - Convert an MNGET page number to a block address

MNCEW2 - Set location Z and the data field for accessing word two of a PCB entry

MNDRVE - Move ENTØRG to last entry in current block and PØINTR to the end of that entry

MNFRE - Set the indicated group of pages in the MNALC list to FREE

MNGET - For the specified number of contiguous pages desired, return the block number of the first page allocated. All blocks in the group are allocated by setting the appropriate word in the MNALC list to NØT FREE. If not enough contiguous blocks were found, -1 is returned.

MNIDLM - Set MNMØD to zero

MNIDLR - Set MNRSL to -1

MNLDDA - Add an entry to the end of MNLDDA (see below).
If a similar entry appears in the middle of MNLDDA, remove it before adding the new entry. Called by MNLØDR after a block is successfully loaded

MNLDDA -	NOT a routine, but the list of blocks currently in core, whether fixed or unfixed. Each one-word entry contains the address of the PCB control entry for the respective block. The last word on the list represents the most recently used block (see MNLDST, etc.)
MNLDDL -	Delete an entry from MNLDDA. Called from several routines, particularly MNDEB after a block is no longer in core
MNLDFD -	Called from MNLDAD and MNLDDL to find a specified entry in MNLDDA
MNLDPD -	Called to compress MNLDDA by one word (delete an entry) from MNLDDL and MNLDAD
MNLDST -	Write out in an IR temporary (as well as free core and delete from MNLDDA) the oldest unfixed block in MNLDDA. Called from MNLDPR when out of space
MNMDAT -	Test if ACC and word at PØINTR match. Uses the absolute value of the difference and masks result with MNRSL
MNMVR -	Move data into DSBANK
MNMVX -	Move data within DSBANK in reverse (e.g., end to beginning). Used to expand blocks and entries
MNNIS -	Called by MNRPL to complete expansion processing. Calls MNREAL to expand block, then moves in remaining data from MNRPL operation
MNNMST -	Resets BLKNUM as specified by ACC
MNSALC -	Set the specified core page to NOT FREE by marking the appropriate MNSALC word
MNSCAN -	Scans each sequential entry in the current block for a data match
MNSTNM -	Same as MNNMST, but uses SVPRM2 instead of ACC

MNTALC - Test if the specified core page is marked
 FREE or NOT FREE in MNALC

MNTRMX - Position ENTØRG and PØINTR to the first
 terminal entry in the current (assumed
 terminal) block

MNXTR - Extract data from DSBANK

MNYXF - Calls MNSCAN to perform a Y, X vicinity test
 on entries in the current block

MNYXMV - Sets SVPRM5 and SVPRM6 in UTBANK from +112 (Y),
 +113 (X) in the current PCB

PCBXGA - Resets PCBADR for the current terminal. Uses
 the PCB copy of BLKNUM (e.g., current block)
 to reset ENTØRG, PØINTR, and BLKØRG

PCBXGT - Calls PCBXGA after fetching current console
 address from GLBANK (INTCØN). Called by
 MNNTRE, MNDPC, DPUPDC

PCBXST - The analog of PCBXGT. Saves current BLKNUM in
 PCB; stores ENTØRG and PØINTR as appropriate
 (correct BLKØRG already in PCB entry)

SADPC - Add word at specified location in PCB to ACC
 (same as GLASP SADPC)

SSTCV - Same as GLASP SSTCV

SSTPC - Same as GLASP SSTPC

UTNCV - Reset console vector address in GLBANK as
 specified by ACC

1.2 Discussion

The operation, and use of, many DSMN routines is obvious and clearly discernible from the listings. Only the more complex and critical routines will be described here.

Loading a data structure block in the current picture is normally straightforward. If the block is already in core, the PCB entry is marked by MNLØDR as fixed and ENTØRG, PØINTR, BLKNUM, and BLKØRG are reset to indicate that the specified block is now the current block. If the block is not already in core, it must be either in an IR temporary or not-initialized. MNLØDR uses the allocated page length in word one of the PCB entry to request space from MNGET. If MNGET returns the first page number of a suitable contiguous area, MNLØDR can complete its processing. The block is brought in from IR (via DSMNGØ) unless the PCB entry is marked not-initialized; noninitialized blocks are never stored in IR temporaries. ENTØRG, PØINTR, BLKNUM, and BLKØRG are set as above, and the proper return code is saved (ØK or not-initialized). Finally, MNLØDR places the PCB entry address for the specified block on the end of a special list, MNLDDA. This list contains a one word entry for each block that is successfully loaded by MNLØDR, with the last entry being the one most recently loaded. Whenever a loaded block is stored in an IR temporary or deleted (MNDEB) from the current picture, its entry in MNLDDA is deleted. The page list in MNGET (e.g., MNALC) plus the cronological in-core block list (MNLDDA) provides a description of core availability and usage in DSBANK at any time.

Note that in addition to the obvious RØUTX call, MNLØDR may be invoked by other circuitous paths. A simple one is via MNIBA; the equivalent of MNDEB is performed, then MNLØDR is called to set up the space in core (e.g., not-initialized is the usual return code of MNIBA). A deeply embedded path is via MNRPL: a block must be expanded to accept inserted data, so MNNIS and MNREAL are called. MNREAL explicitly stores the block (MNSTØR) in an IR temporary and alters the page allocation in the PCB entry. Then MNREAL calls MNLØDR to bring the block back into core with the new space allocation.

The problem arises, of course, when MNGET does not locate enough space. MNLDDA now comes into use; MNLØDR calls MNLDST to swap-out an in-core block. MNLDST scans the list of blocks to find the least recently used block that is unfixed. Fixed blocks, by definition, cannot be swapped-out. If no unfixed blocks are found, a lockout condition has occurred; MNLØDR must pass the appropriate return code. If an unfixed block is found, its entry is removed from the list, and the block is written into an IR temporary via DSMNG⁴. The core occupied by the block is then released via MNFRE, and MNLDST returns to MNLØDR. MNLØDR again tries to load the desired block (e.g., calls MNGET for space) and, if successful, processing terminates as above. If the block still cannot be allocated, MNLDST is called again. This continues until either the block is loaded or MNLDST runs out of unfixed blocks.

MNRPL is a very general routine for altering data within the current data structure block, as is shown by the parameter descriptions in the listing. MNRPL first calculates the length of data to be removed and stores this value in MNRPL⁴; a simple test ensures that the length does not exceed the size of the current entry. The displacement from ENTØRG to PØINTR is saved in MNRPL⁵; this value is used to reset PØINTR after the operation. The difference between the lengths of the data to be removed and the data to be inserted is saved in MNRPL⁶; this is the amount that the entry (and block) must be contracted or expanded. MNMØV is then called to move in all of the input data (MNRPL⁶ is positive, removal len > insert len) or the first part of the insert data (MNRPL⁶ is negative, removal len < insert len; MNRPL⁴, the removal length, is used instead of SVPRM⁷, the insert length, for the call to MNMØV). If MNRPL⁶ is in fact zero, the replacement was exact, so a quick exit is taken. Otherwise, PØINTR and SVPRM⁶ are incremented by SVPRM⁷ (e.g., the TØ and FRØM addresses, respectively, now reflect the MNMØV operation), and MNRPL⁶ is checked again. For MNRPL⁶ positive, the block only needs to be contracted. After some setup, the block length and entry length are altered, and MNMVR is called to move up the rest of the block to the current PØINTR. Finally, PØINTR is reset using MNRPL⁵ to its original position relative to ENTØRG, and MNRPL exits. For MNRPL⁶ negative, the

block must be expanded and the remaining insert data moved in. MNNIS is called for this; after expanding the block allocation with a call to MNREAL, MNMVX is used to move the end of the block down to accommodate the rest of the insert data. The block length and entry length are increased appropriately, MNMOV is called to bring in the data, and MNNIS returns to MNRPL. MNRPL resets PØINTR and exits.

The only hooker in the above procedure is the call to MNREAL. As will become apparent in the following description, processing in MNREAL may be extremely lengthy and might involve a great amount of disk accessing. Keep in mind that MNRPL, even more so than MNLØDR, may be entered via a large variety of paths. MNINE, MNNNE, MNILE, and MNNEN, among others, all call MNRPL. One of the potentially worst cases occurs when MNRPL is called by MNBTB; both blocks must be in-core and fixed during the entire procedure. In fact, as the two blocks in question approach maximum size (1/2 bank each), the likelihood of a lockout increases rapidly, since even a small fixed block from any other terminal could fragment core.

To expand the allocation of a block, MNREAL first tries to add free core pages to the end of the block. MNREAL calculates the page number of the core page just after the block; MNTALC is called, and if the page is FREE, MNREAL calls MNSALC to set the page NOT FREE. The block allocation in the PCB entry is incremented by one, and the number of extra pages needed is decremented by one. If more pages are still needed, the above process is repeated. If enough pages can be added in this way, MNREAL will take a quick return to MNNIS. However, if MNTALC finds that the requested page is already marked NOT FREE, the block cannot be sufficiently expanded in this simple way. So MNREAL saves the displacements of PØINTR, ENTØRG, SHDPTR, and SHDERG from BLKØRG. When the block is relocated in core, these displacements are used to reset the pointers to their original positions within the block.

To relocate the block, MNREAL first calls MNSTØR to write the block into an IR temporary. The first word of the PCB entry is then altered to contain the entire block allocation needed plus bits indicating not-in-core and initialized. MNLØDR is then called to bring the block in

with the new space allocation. Note that the MNSTØR freed the space currently used by the block as well as caused its entry in MNLDDA to be removed. If MNLØDR is lucky, MNGET will find enough contiguous core in some other part of DSBANK for the new block. Otherwise, MNLDST will be called to start swapping-out unfixed blocks. Obviously, overhead for this particular MNRPL operation has become quite high at this point!

When MNLØDR successfully loads the block, MNREAL resets the pointers using the stored displacements and calls PCBXT to copy these new values into the PCB entry. A return is then made to MNNIS. If MNLØDR was NOT successful, this unfortunate event must be filtered all the way back to the routine(s) that initiated the operation.

For working internally with IR temporary files, DSMNGØ and DSMNG4 call DSMNG6 to set up an IR parameter list; DSMNG3 is then called to perform the actual IR linkage. Note that if DSMNG4 was called by MNLDST after a call to MNLØDR from MNREAL during MNRPL processing of a MNBTB operation...things are really tied up! And they will get worse! Not only is core held by fixed blocks and DSMN as a service is busy, but IR as a service will also be tied up during DSMNG3 processing. Fortunately, the CPU and disk are quicker than the terminal users (usually).

DSMNG3 saves the terminal's status within its own save areas during IR processing; since DSMN as a service is busy, DSMNG3 cannot be re-entered, so this saved information will not be wiped out. Unfortunately, DSMNG3 cannot possibly save all of the intermediate routine returns and temporary locations; hence, there is no alternative to DSMN remaining busy as a service. The areas saved are:

1. Current block pointers (via PCBXST)
2. UTBANK SVPRM1-8
3. Console vector SVPRM1-8
4. Console vector RØUTX return data

A call to DSMNIR in GLASP is then made; DSMNIR issues a priority WAIT request (GETIR) and returns when IR is free. DSMNG3 moves the IR parameter list formed by DSMNG6 into SVPRM1-8 in GLBANK and calls IR

via RØUTX. Areas 3 and 4 saved above would obviously be wiped out by the RØUTX call to IR; while IR is in operation, DPU servicing or other direct routine calls could easily wipe out the information in areas 1 and 2.

When IR processing is complete, ØPDØNE will return to DSMNG3; the RØUTX call is, for all external appearances, quite normal. DSMNG3 resets the current console (INTCØN) in GLBANK and restores all four areas previously saved. The complete status of the terminal and DSMN is now the same as before the IR call. DSMNG3 returns and DSMN processing continues.

Creating the initial form of a mnemonic storage block (MSB) from the current picture is the tedious task performed by MNMNEM. The only input to this routine is the Y, X values needed to position the parameter model list, plus, of course, the current picture. MNMNEM first allocates (via MNIBA) a two page work area in block 12₈ and initializes this new block via MNIBD as follows:

+0	24	Total block length
+1	11	First entry length (control entry)
+2	0	Space for six character name
+3	0	
+4	0	
+5	4040	Default type word (name.0)
+6	0	Space for Y and X when used as
+7	0	an instance
+10	0	Space for instance sequence number
		when used
+11	0	Space for creator's ID
+12	3	Second entry length (param model entry)
+13	Y	Position of the parameter model list
+14	X	as specified by input
+15	3	Third entry length (type-name entry)
+16	1344	Y, X of top menu line
+17	0	
+20	1	Fourth entry length (terminal list entry)
+21	3	Fifth entry length (line entry)
+22	0	Line entry displacements
+23	0	

The above is functionally complete. If the current picture has an empty type-name list, no external terminals, an empty line block and an empty text block, saving the above data--even if the calling program does not insert the six character name, the ID, or call the text editor to fill in the parameter model entry--will result in a usable (though very uninteresting) mnemonic. Note that when an MSB is used to create an instance in a picture (e.g. "integrated" by MNMNIN), the first three MSB entries are copied bodily into the picture's instance block; the terminals in entry four are inserted into the picture's terminal block. In both these cases, MNMNIN modifies certain Y, X values in the copied data; these modifications are described in the discussion of MNMNIN below. Whenever a picture's instance block is displayed, the line and text portions of each referenced constituent instance are recreated from the respective MSB. In this case, certain displacements are added to the Y, X values in the line (entry 5) and text (any subsequent) entries as display file formatting is performed. This is also discussed in MNMNIN below.

Consult Figure 11; after forming the above "empty" mnemonic, MNMNEM proceeds to fill in data whenever available from the current picture. If certain data is not available, the corresponding entry in the MSB remains empty but functionally complete. Entry one is already complete; the calling program is responsible for filling in the name, the type word, and the creator's ID before calling SVENMN to add the MSB to the library. The Y, X and instance sequence number locations are not used until MNMNIN integrates the MSB as an instance. Entry two is also already complete; the calling program is responsible for calling the text editor so that the terminal user may enter the desired parameters. If no such call is made, entry two is just a text entry with no lines. Setting the third entry is a good demonstration of the block-to-block facility; MNMNEM proceeds as follows.

The MSB (block 12) is the current block; it is in-core and fixed. MNPERG is called to position ENTØRG and PØINTR to the third entry;

MNBSV is then called to save the current status (e.g. ENTØRG and PØINTR) of the current block (No. 12) in the appropriate PCB entry (PCB+60). MNLØDR is called to load the terminal block (block 4); this makes block 4 the current block. If lockout occurred during MNLØDR, a quick exit is taken and the calling routines must check for this return code; the terminal user should simply try the operation again. If block 4 is not initialized, there can be no type name list, so entry three of the MSB will remain as it is. Block 4 is set unfixed, block 12 is left in its "saved" condition, and MNMNEM proceeds to load the line block (No. 0); see below. If block 4 is loaded successfully (e.g., it is initialized), MNBSV is called to save its status, as was done for block 12. Note that MNLØDR always sets ENTØRG and PØINTR to the first entry in the loaded block; since the first entry in the terminal block contains the type name list, this is the entry that will be used to set the third entry in the MSB. Hence, MNPERG was NOT called before MNBSV, since the desired entry was already pointed to. MNBRS is now called to make the MSB, block 12, the current block again; the appropriate pointers are reset from ENTØRG and PØINTR in the PCB+60 entry. The MSB with its pointers set to the third entry, is the TØ block, while block 4 with its "saved" pointers set to the first entry is the FRØM block. MNBTB is now called to "replace" the data in the current TØ block location with a copy of the data at the FRØM block location. Obviously, since the MSB may be expanded by subsequent calls to MNREAL from MNRPL, the lockout condition must be tested when MNBTB returns to MNMNEM. If lockout occurred, a quick exit is taken, as above. If the operation completed normally, MNMNEM proceeds to work on MSB entry 4, the terminals. Note that even in the case of normal completion, the MSB might now be in a different core location; the pointers, ENTØRG and PØINTR, were properly reset to the necessary relative locations within the block by MNREAL.

Processing the other entries in the MSB is handled in the same fashion. The pointers to the desired MSB entry are set, and the MSB is "saved" with MNBSV. Then the desired picture block is loaded, its pointers set, and its status saved with MNBSV. The MSB is made the current block with MNBRS, and MNBTB is called to transfer the data. At all

appropriate points, lockout is tested. Uninitialized blocks are checked for, and all blocks are unfixed as soon as possible.

After successfully setting the type name entry in the MSB, MNMNEM calls MNNXTE to set the pointers to the fourth (terminal's) entry; the MSB status is then saved with MNBSV. Since the terminal block, No. 4, is already loaded, MNBRS is called to make it the current block. Its pointers are then moved to the first terminal entry, and MNMNEM proceeds to extract each external terminal and insert the appropriate terminal data, if any, into the MSB. Finally, all terminals are checked, MSB entry four is complete, and the terminal block is unfixed. MNMNEM is now ready to process the picture's line and text blocks.

The first (only) entry in the line block (No. 0) is copied into the MSB's fifth entry in a manner directly similar to the processing of the type name entry described previously. Then each entry, if any, in the text block (No. 1) is added as a new entry at the end of the MSB. In this case, MNBTB is called so that MNNNE is used instead of MNRPL.

The MSB is complete; MNMNEM processing terminates.

As mentioned in MNMNEM, the MSB is used (1) to form an instance in the current picture's instance and terminal blocks, and (2) for displaying the line and text portions of the instance during REGEN. When MNMNIN is entered, it assumes that the MSB to be integrated into the current picture is already loaded in block 12₈; processing is essentially carried out as in MNMNEM. That is, MNBSV, MNBRS, and MNBTB are used to copy information from the MSB to the instance and terminal blocks; the same type of error checking (lockout, not-initialized, etc.) is also used. The only input to MNMNIN is the Y, X position of the instance center. Note that when the MSB was constructed, the locations of the terminals, the parameter model list, the lines, and the text entries were all stored as absolute screen coordinates. However, when displayed as an instance, this information must be displayed relative to the instance center. To facilitate this adjustment, the MSB is assumed to have been created with its center at (1000, 1000). The adjustment, then, consists of setting

$$YD = Y + (YN-1000)$$

$$XD = X + (XN-1000)$$

where YD and XD are the adjusted coordinates for the instance, Y and X are the coordinates stored in the MSB, and YN and XN are the input coordinates of the instance center. Consult Figures 9 and 12.

MNMNIN builds the control entry (first of three entries to be added to the instance block) from the data in the first MSB entry, the input Y, X, and the instance sequence number (INST). INST is obtained by adding one to the INST found in the last instance (call to MNDRVE followed by call to MNPERG plus a call to MNPLØD). If the instance block has not previously been initialized, it is initialized, and zero is used for INST. The control entry is added to the instance block with MNNNE.

The parameter model entry is added to the instance block next; then the entry's Y, X is adjusted to YD, XD. The type name entry is the last of the three entries to be added to the instance block; its Y, X is not altered since it is always displayed in the menu area if used. The instance block is now set to unfixed, and the terminal block is loaded.

If the terminal block was not initialized, it is initialized now; zero is used for the first terminal sequence number (TRM). Each terminal in the MSB's fourth entry is extracted; the Y, X is altered to YD, XD; the next TRM is obtained with a call to MNHTN; and the resulting entry is added to the terminal block. This continues until all terminals are extracted. Then the terminal block is marked unfixed; block 12 is left as the current block; and MNMNIN processing is at an end.

During display of the line and text entries of an MSB, DPUM uses the YN, XN obtained from the instance control entry for the instance to modify the appropriate Y, X fields. Consult DPUM in the listing. The terminals and parameter model list, of course, are already set to the proper locations due to MNMNIN's manner of integration.

2. DPU Routines

Entries

DPUENQ - RØUTX call entry
 DPUPDC - Direct call entry

RØUTX Calls

DPUENQ - Immediate operations; enqueue DPU

Direct Calls ("Add data to the current display file" = AD)

DPUA - Display a text-line
 DPUB - Display a line
 DPUC - Display an arbitrary preformed file or display
 the current display file just built
 DPUDI - AD to move the beam invisibly (blanked)
 with a vector
 DPUDP - AD to move the beam visibly or invisibly with a point
 DPUDV - AD to move the beam visibly with a vector
 DPUE - AD to display the specified text-line at a
 specified Y, X
 DPUF - AD from \leq five words from SVPRM4-8
 DPUG - AD of any arbitrary length
 DPUH - AD consisting of a specified vector block
 DPUI - AD formatted from the text-lines of the
 current (assumed text) entry
 DPUJ - AD formatted from the text-lines of all entries
 in the current (assumed text) block
 DPUK - Uses DPUKA and DPUKB to send the specified text-
 line to the terminal's character line buffer
 DPUL - AD formatted from the terminals and terminal
 connections of the current (assumed terminal) block
 DPULY - AD formatted from the specified terminal entry
 DPUM - AD formatted from the current (assumed mnemonic
 storage) block
 DPUNL - AD formatted from the point groups of the current
 (assumed line) block

Internal Routines

DPUDQ -	After display file complete (and being transferred by DPU), link to DPUDQ1 in GLASP to await DPU completion. DPUDQ1 sets WAIT priority call GTDPU; then returns control to segment via XCTL2 (immediate command) or XCTLS (direct call) as appropriate
DPUFER -	Save specified reference center input
DPUHBN -	Turn high (bit 0) ACC bit on (=1)
DPUHL -	Add an exit mode sequence (0000, 4000) to the display file
DPULEN -	Check that enough space is left in the display file buffer for the next insert
DPULW -	Display terminal connections; all or just visible connections can be specified
DPULX -	Display terminal symbols and/or characters as specified
DPUM4X -	Called from DPUM to display the line entry of a mnemonic storage block
DPUONE -	Add the single word in the ACC to the display file
DPUPNT -	Increment PØINTR by 1 and load the ACC from the word at that location
DPUQ -	Increment or decrement the display file pointer by the ACC amount
DPURAD -	Reset the display file pointer to the beginning of the buffer and set an initial command (if SVPRM8 not zero)
DPUREF -	Reset the reference center with the saved data (see DPUFER)

All DPU routines perform well-defined functions in a straightforward manner. The necessary parameter list and a short description precede each routine in the listings.

APPENDIX E

CALR (CALR1):0	p. 120
GSAM (GSAMV1):1	p. 123
GUT1 (TEXTN):30	p. 125
GUT2:31	p. 129
GUT3 (REACT):32	p. 133
DEX1 (DEXCHK):33	p. 138
PSIR (IRINIT):37	p. 139

CALR (CALR1):0

CALR1 (*4000)

System Entry

CLEAR CONSOLE
VECTOR AND PCB
TO ZERO.SET
DEVICE ENTRIES.
SET FIRST DISPLAY

1

SYSL
SEND DISPLAY TO
TERMINAL (CALOUT)

RELESE
(Await Input)

A

B

C

DPU TIMER (Program Pick)
Keyboard (Loggon)
(Logon)

B

CALTM1

SAVE MENU BLK
IN IR (SVNMBK)

B1

CALTM3
ERASE SCREEN (DPU)
DELETE CORE COPY
OF MENU BLK (MNDEB)
SET TERMINAL
INACTIVE

PRINT LOGOFF
TTY MSG (LOG)

RELESE
(Exit from CALR)

CALR2 (*4200)

Called Program Return

CLEAR CONSOLE
VECTOR AND SAFETY
DELETE (MNDEB)
OF ALL BLOCKS.
CLEAR PCB

2

A

CALDP1

SAVE FIRST
SIX KEYBOARD
WORDS (FREDPU)

CALL DEX1 TO
CHECK LOGON STRING

NO

CALSRX

FOUND
IN DIRECTORY

YES

3

CALSRX NO

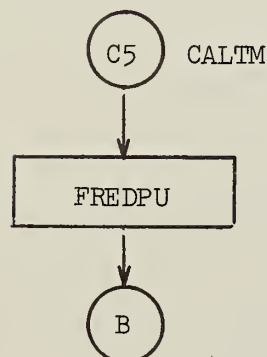
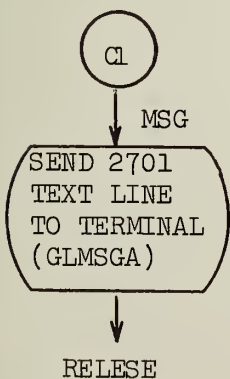
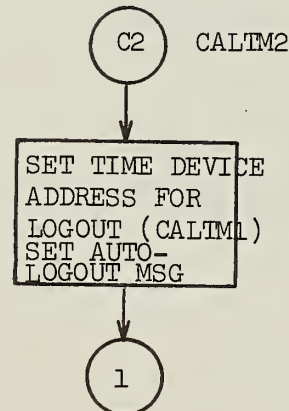
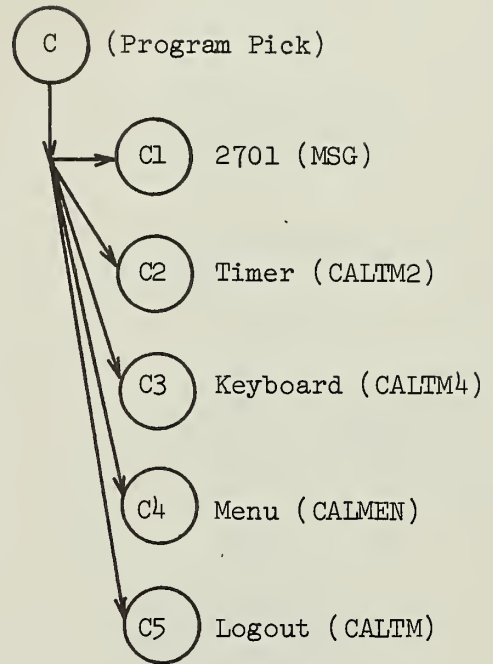
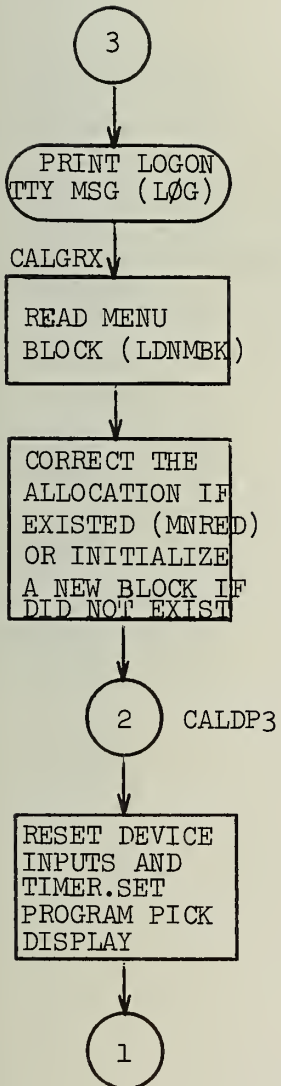
PRINT THREAT
TTY MSG (LOG)

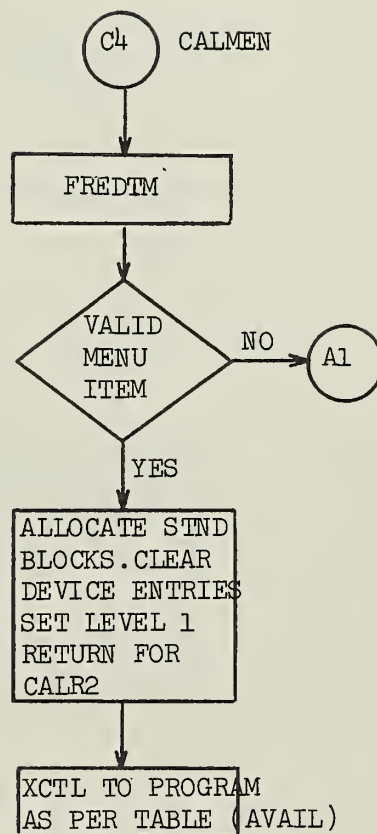
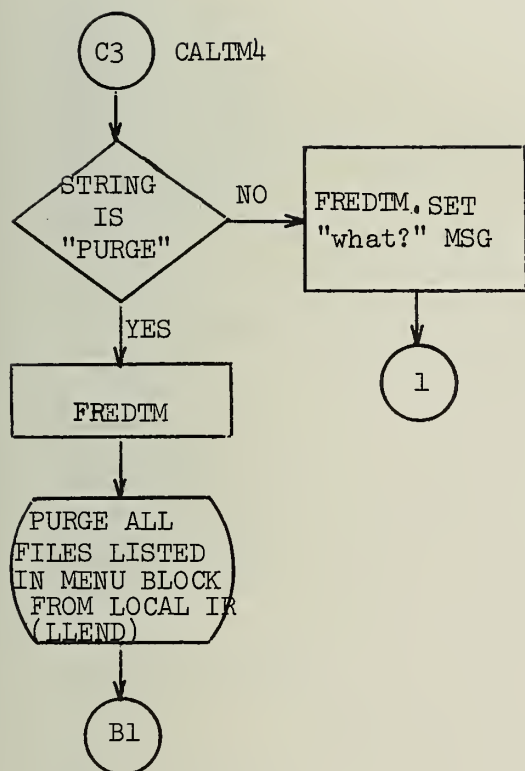
A1

CALNØN

SET NOT
FOUND MESSAGE

1





GSAM (GSAMV1):1

GSAMV1 (*4000)
Level 1 Entry

SET CONSOLE
VECTOR (STOP)
FOR MAIN MODE.
ENQUE DPU.ADD
MAIN MODE TEXT
IN MENU AND
PROMPT AREAS

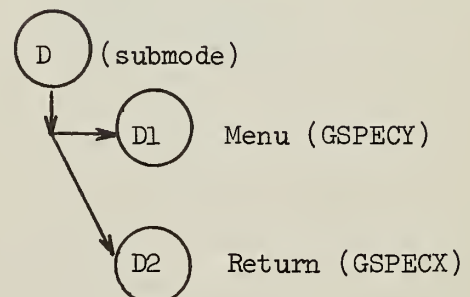
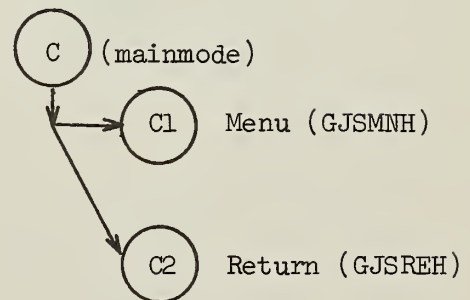
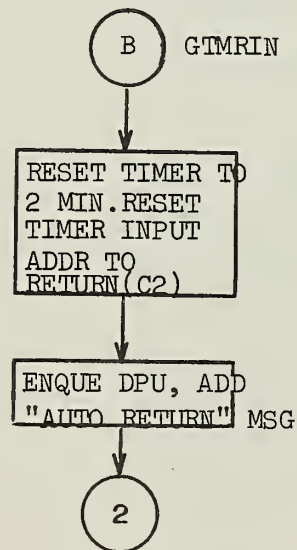
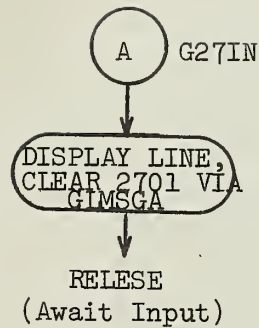
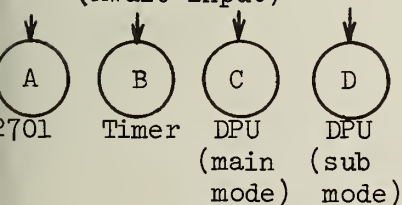
1 GSAM3

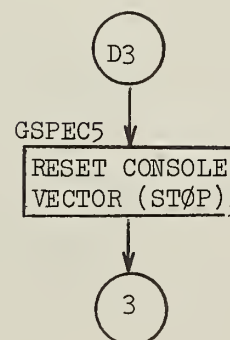
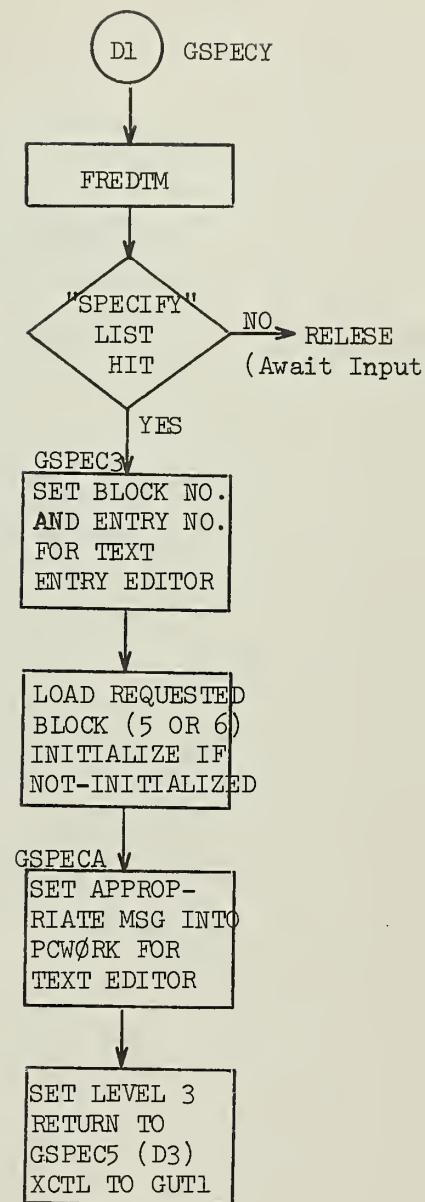
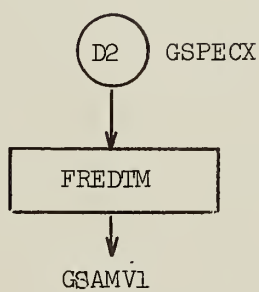
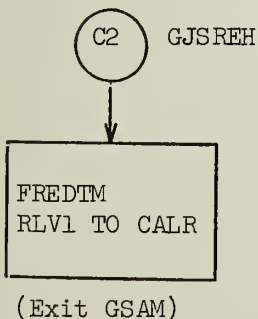
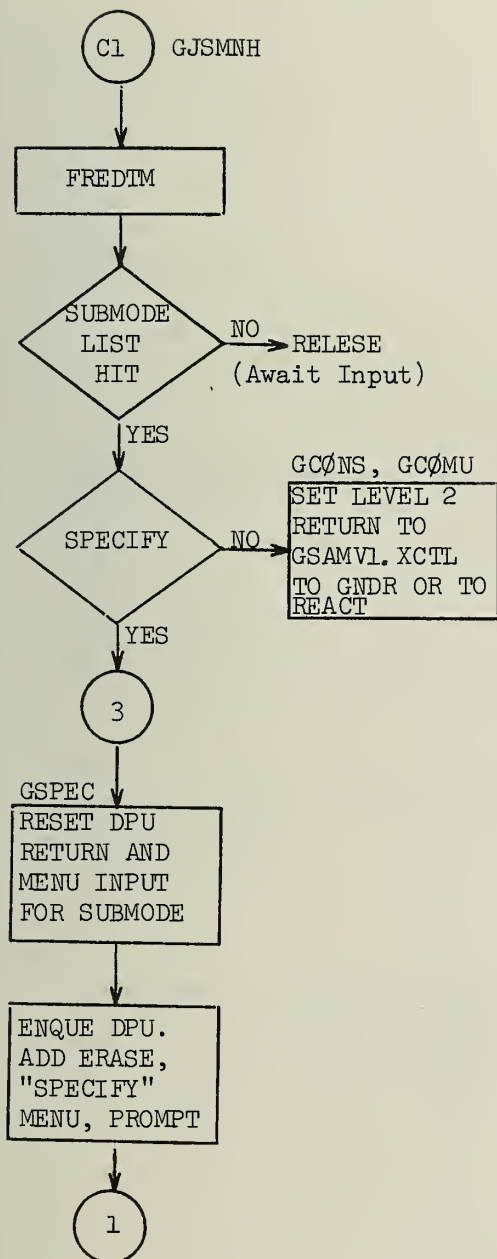
ADD FRAME,
SYSTEM LINE,
USER LINE,
RETURN, PANIC

2 GSAM2

SEND TO
TERMINAL
(GREGP2, DPUPDC)

RELEASE
(Await Input)





GUT1 (TEXTHN):30

TEXTHN (*4000)
Level 3 EntryCLEAR
GREGEN FLAG

1

REGEN

LOAD INDICATED
BLOCK TO BE
MODIFIED. ENQUE
DPU. ADD ALL
FRAME TEXT.
ADD INDICATED
ENTRY FROM BLKUNFIX BLOCK.
ADD MSG FROM
PCWORK. SEND
DISPLAY TO
TERMINALGREGEN
FLAG ON

YES

DO REGEN

2

RESET CONSOLE
VECTOR (STOP)

D1

D1

APPEND

PASS APPEND MODE
KEYBOARD ADDR

3

SET KEYBOARD
INPUT ADDRESS
CLEAR HITS
FLAG

RELEASE

(Await Input)

A

B

C

Timer

2701

DPU

A

PØPT

RLV3

(Exit GUT1)

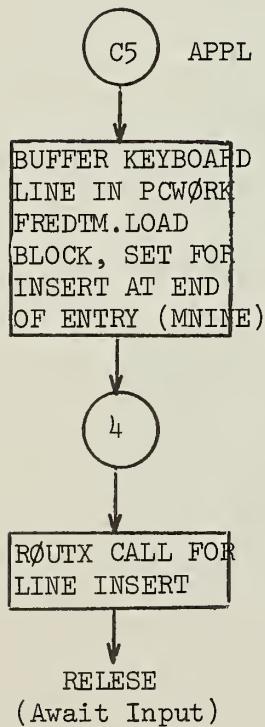
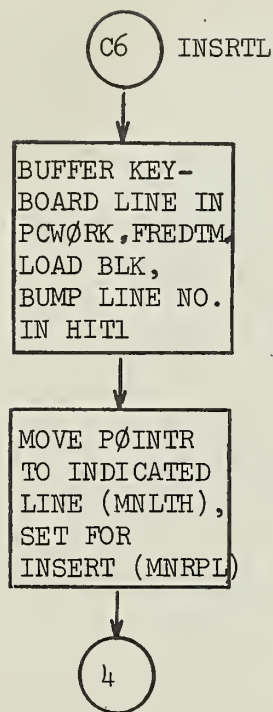
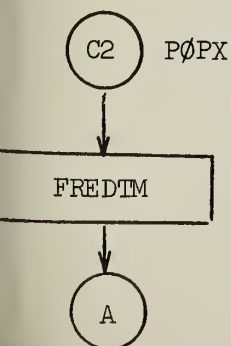
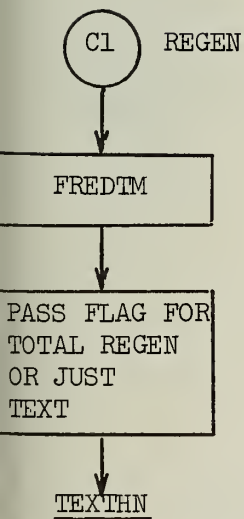
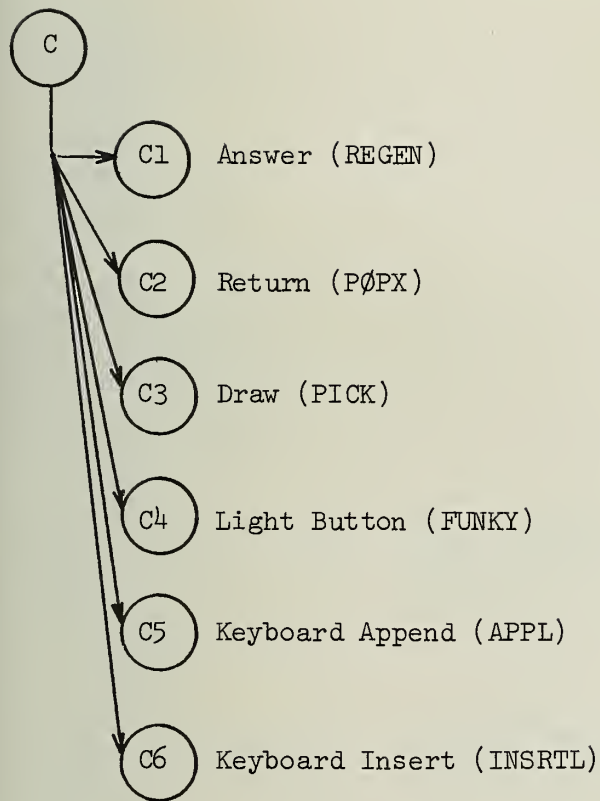
B

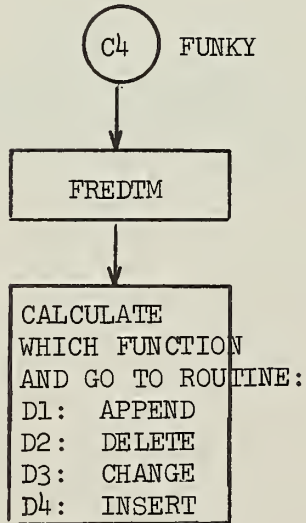
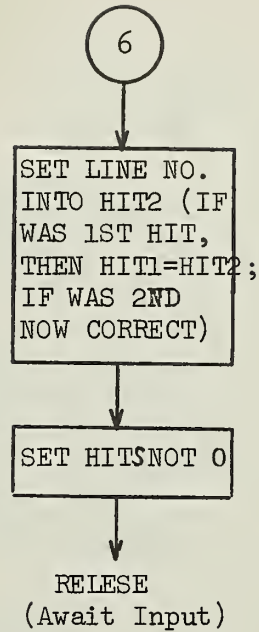
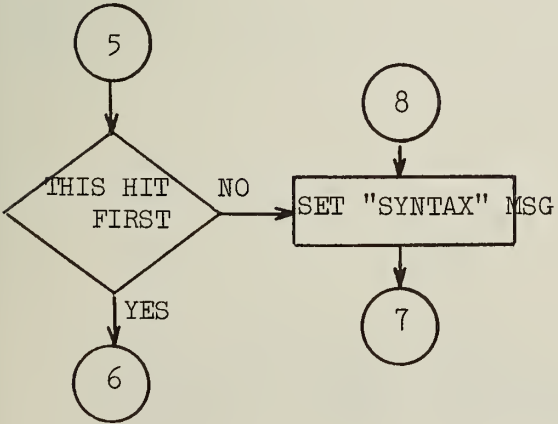
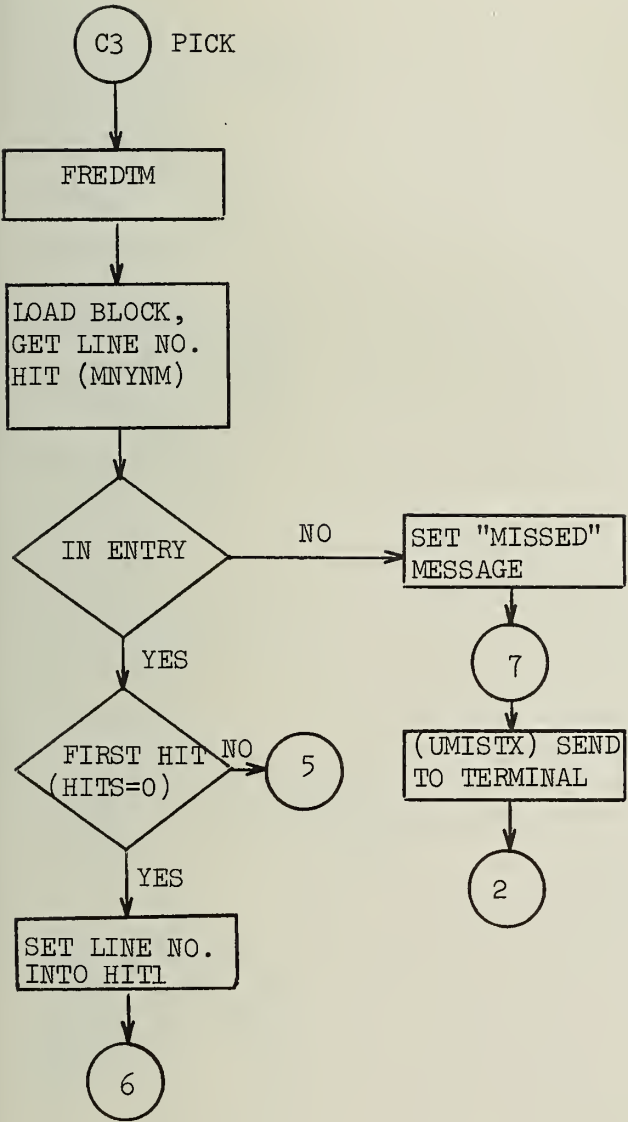
CØMIN

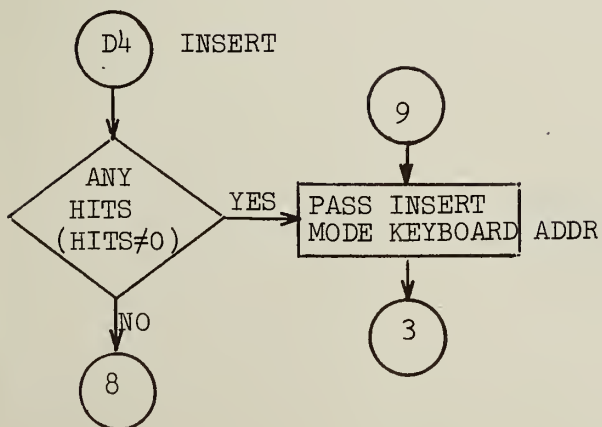
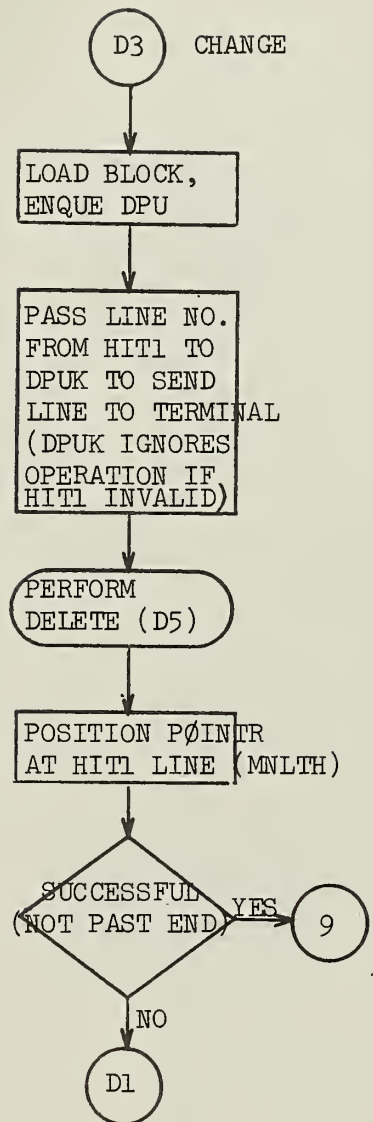
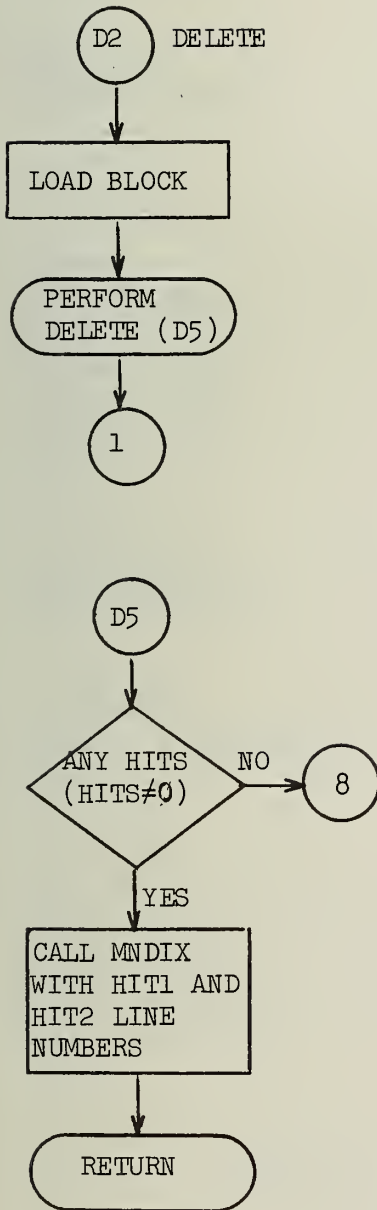
DISPLAY LINE,
CLEAR 2701
VIA GLMSG

RELEASE

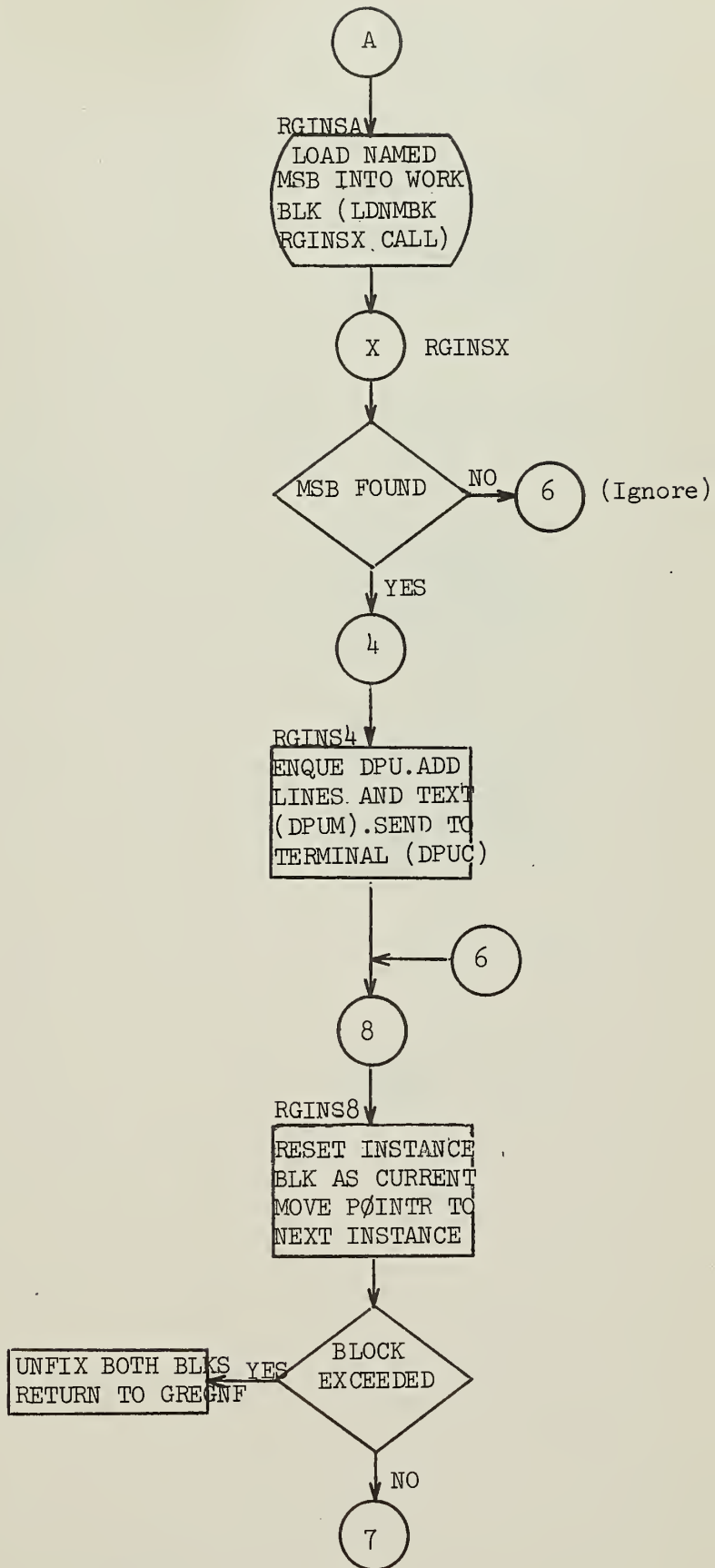
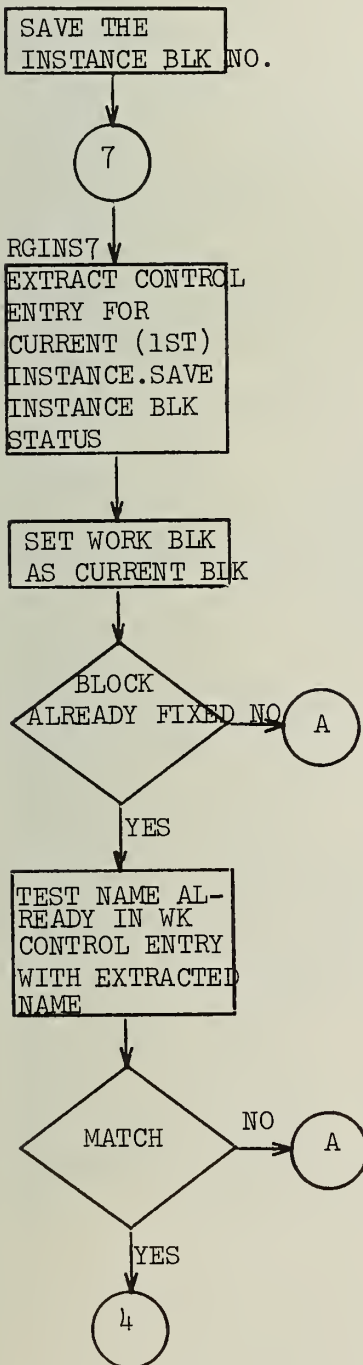
(Await Input)

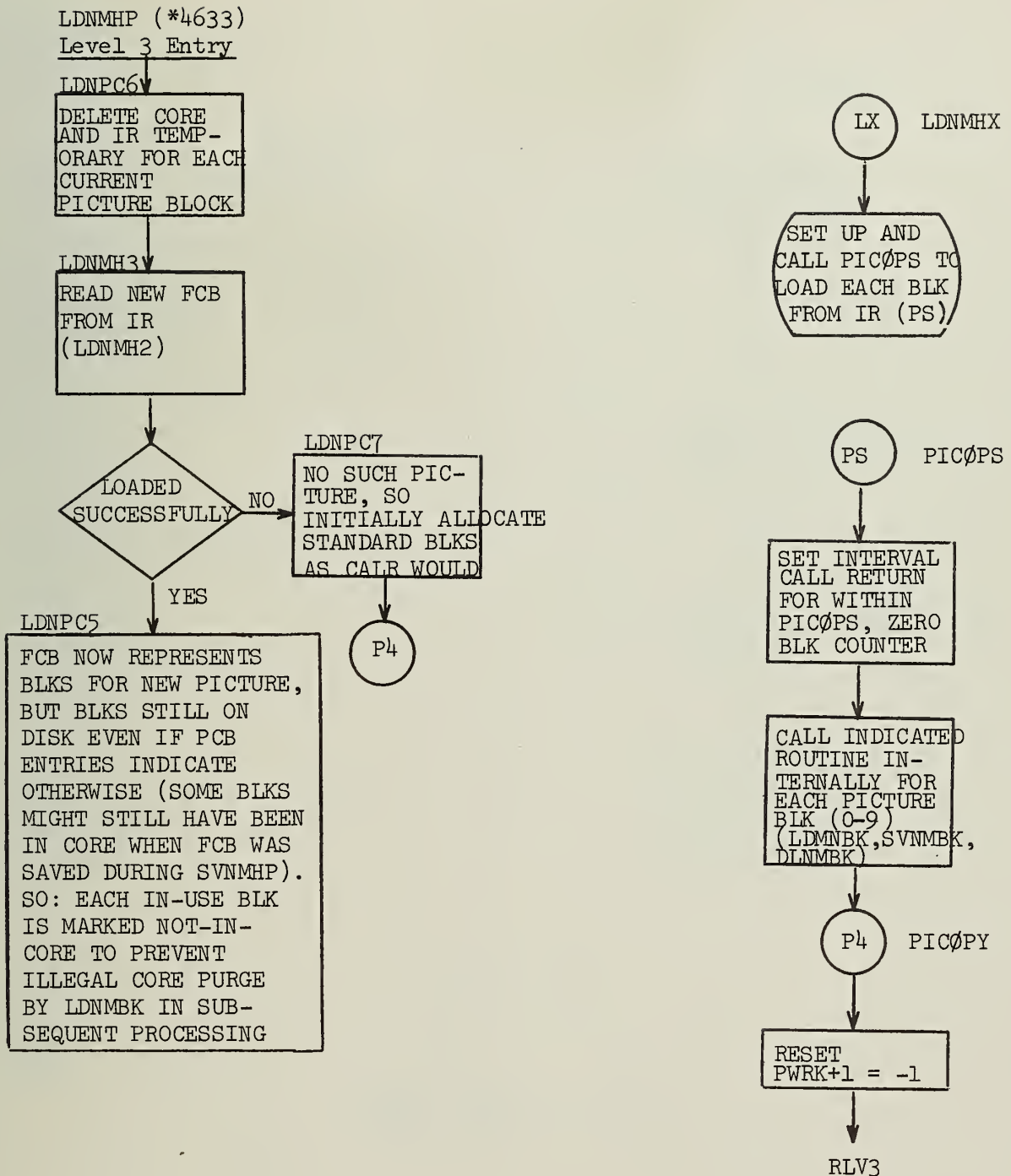




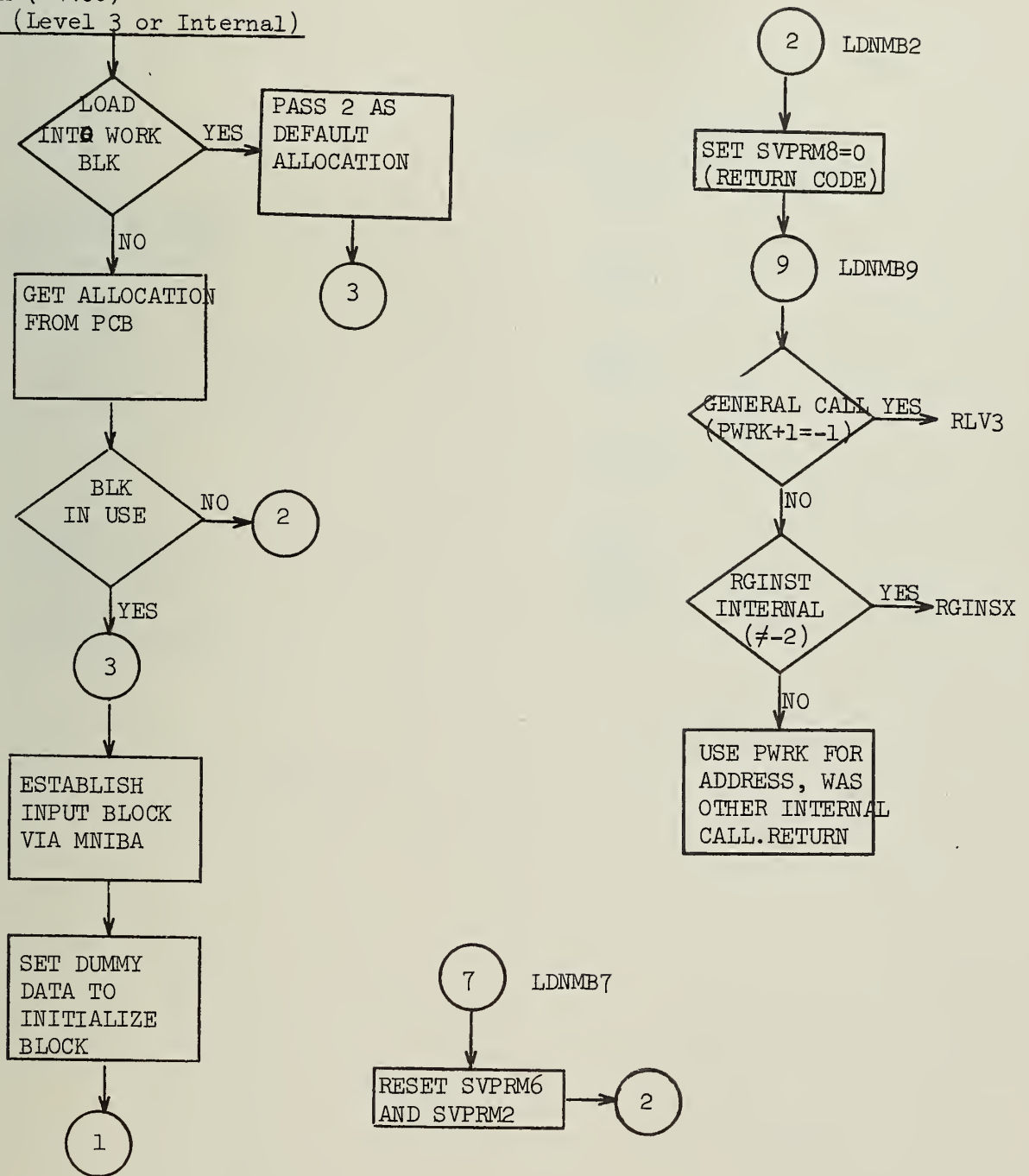


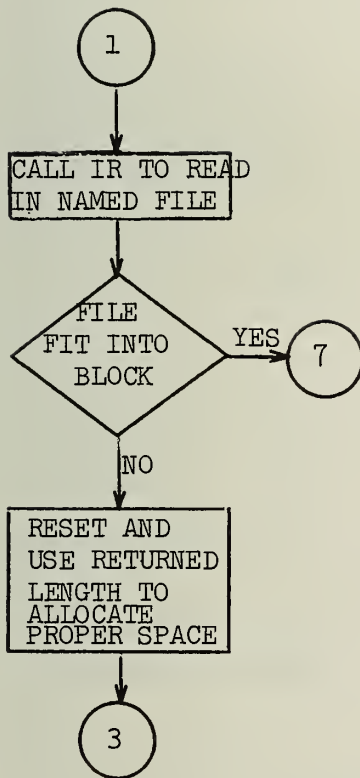
RGINST (*4000)

REGEN Entry



LDNMBK (*4400)

Entry (Level 3 or Internal)



Routines NOT Included

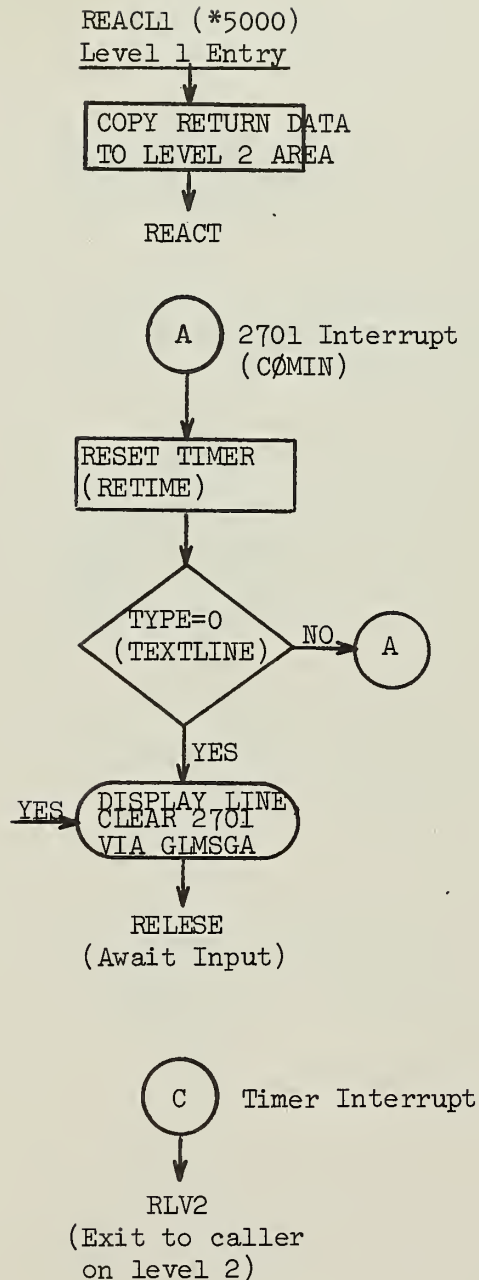
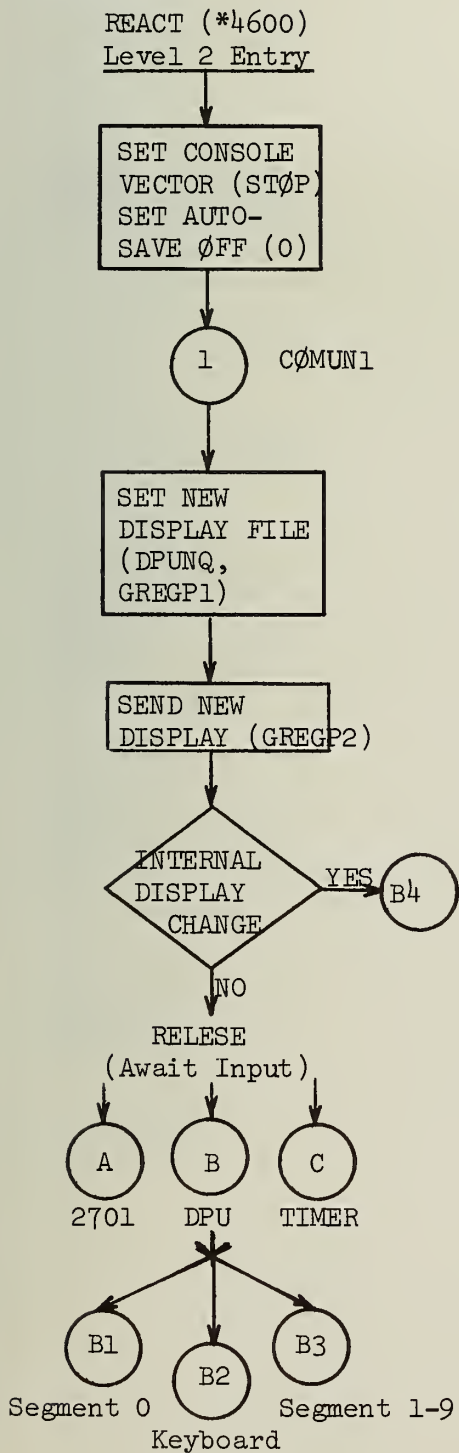
SVNMBK
DLNMBK

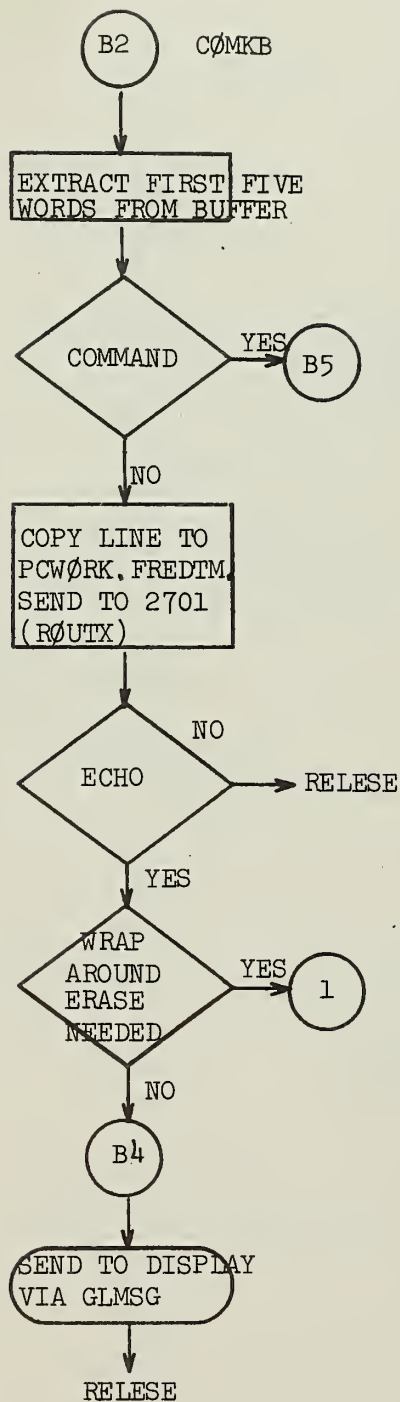
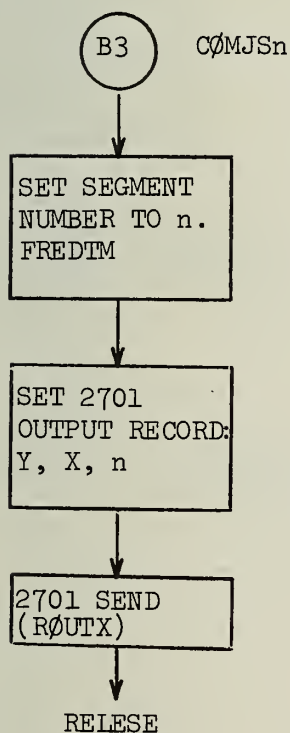
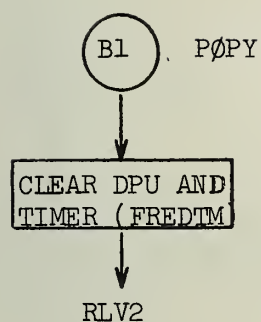
SVNBHP
DLNMHP

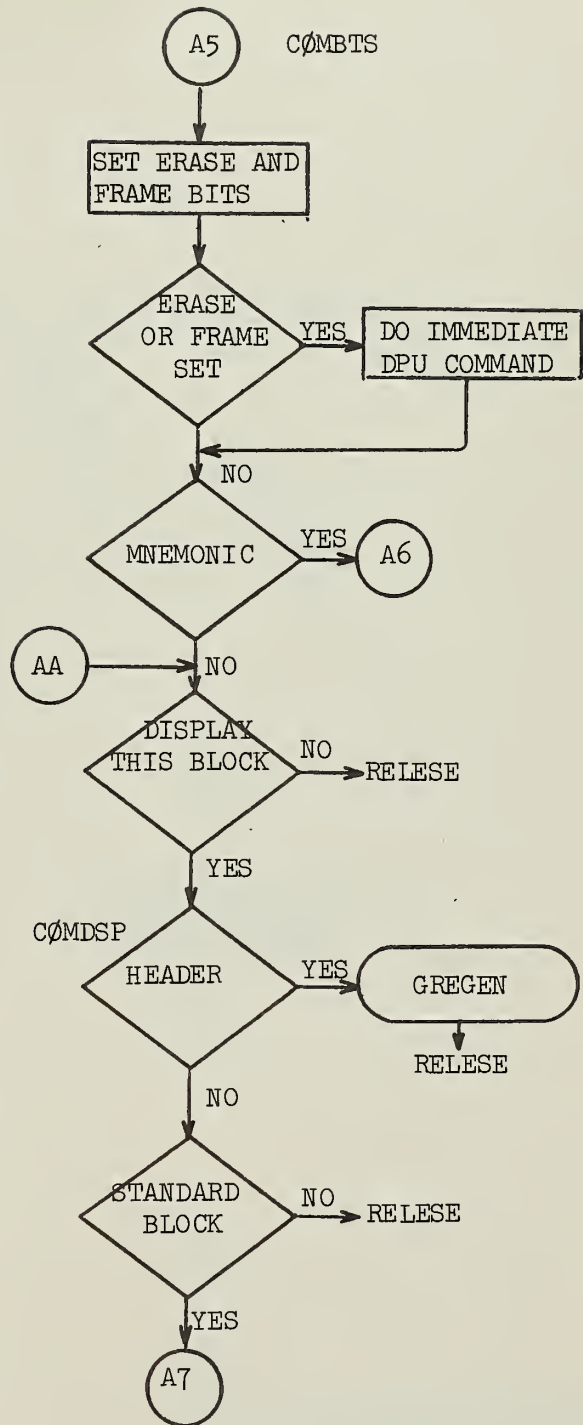
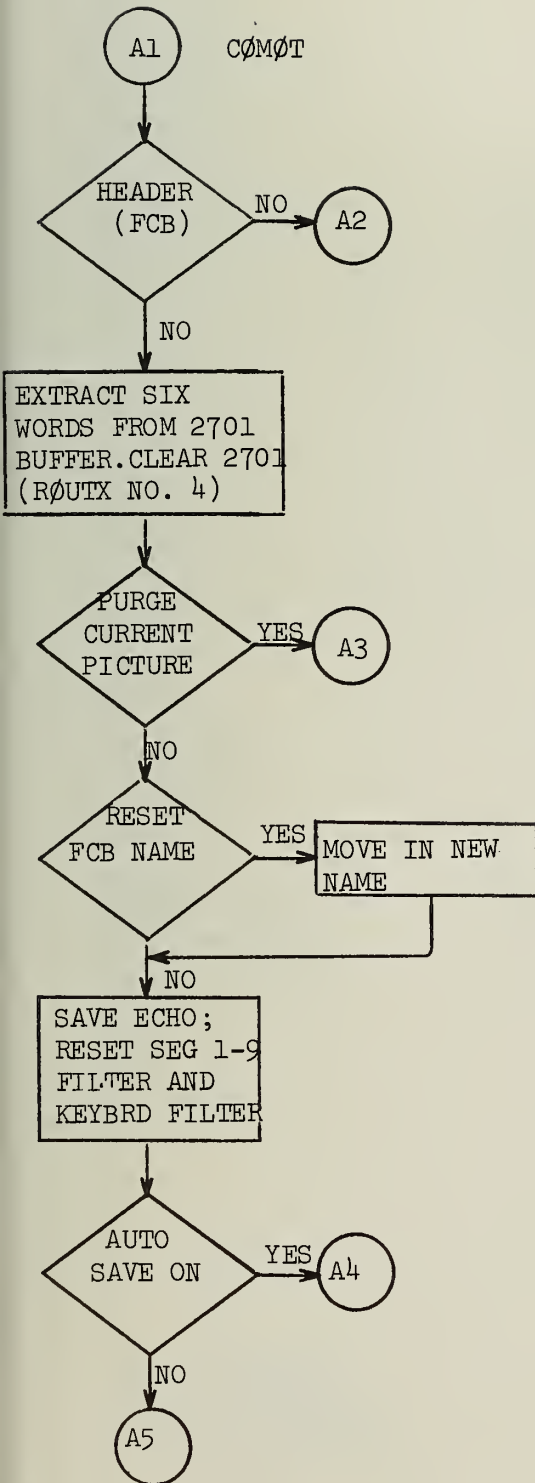
SNRPC
SNDBLK
SNDHDR

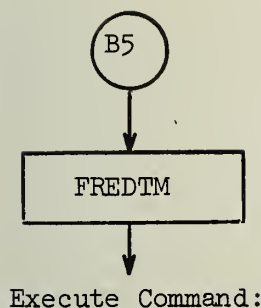
SVENMN
DELNMN

MENUD
MENUA
MENUX
DIMENN

GUT3 (REACT):32







!! (EXIT) → RLV2

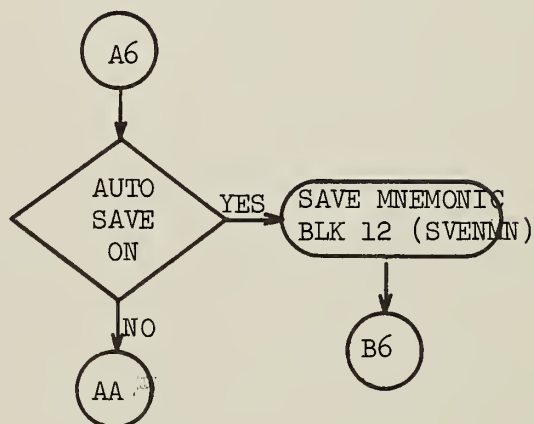
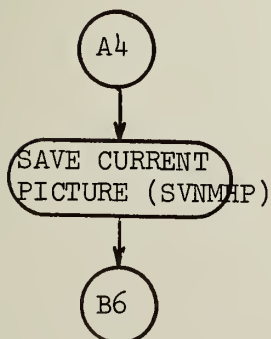
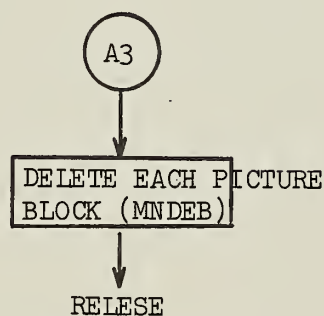
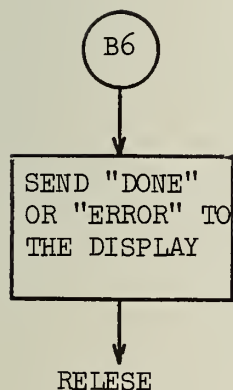
!" (XCTL to GNDR, level 1 and 2 returns remain as set)

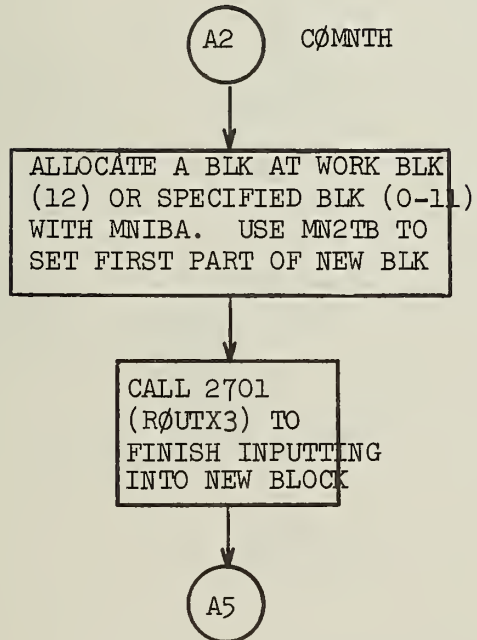
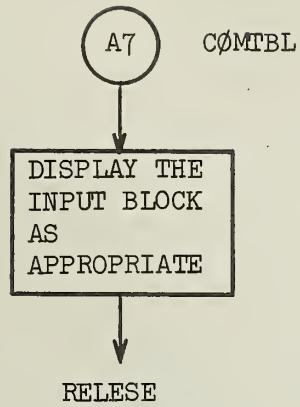
(Load/Send named picture) → B6

&& (Load/Send named mnemonic) → B6

C# (Send current picture as is or renamed) → B6

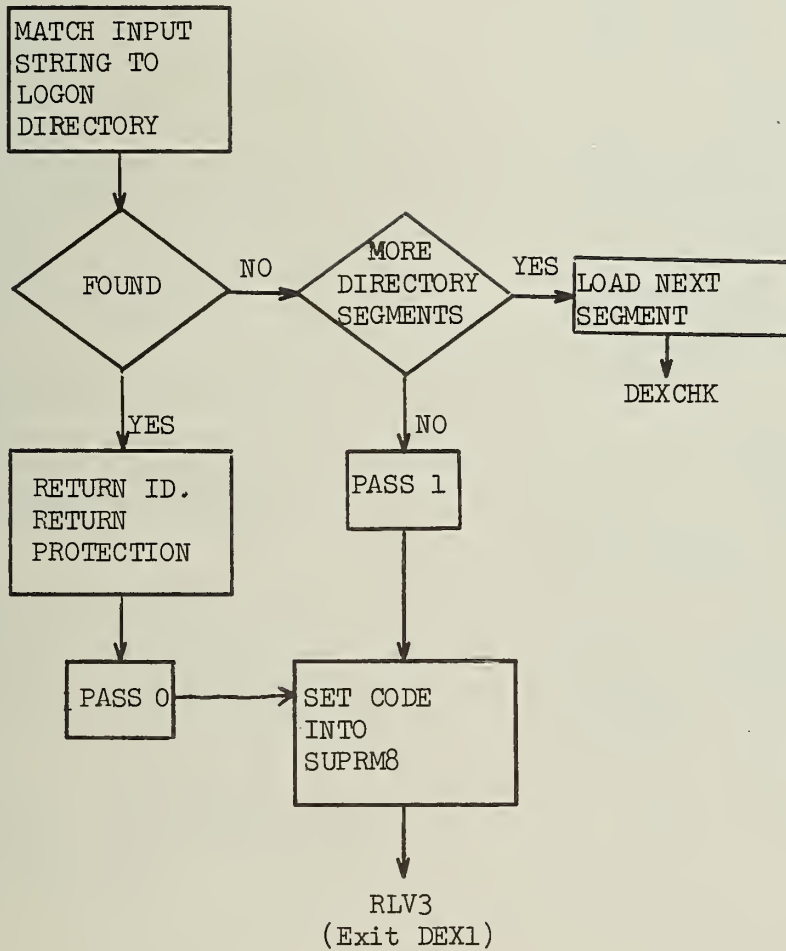
C& (Send current mnemonic as is or renamed) → B6



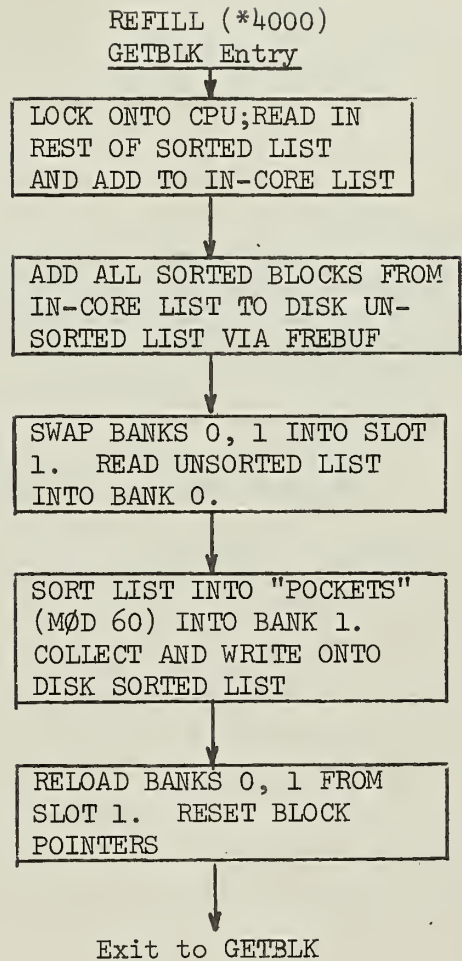
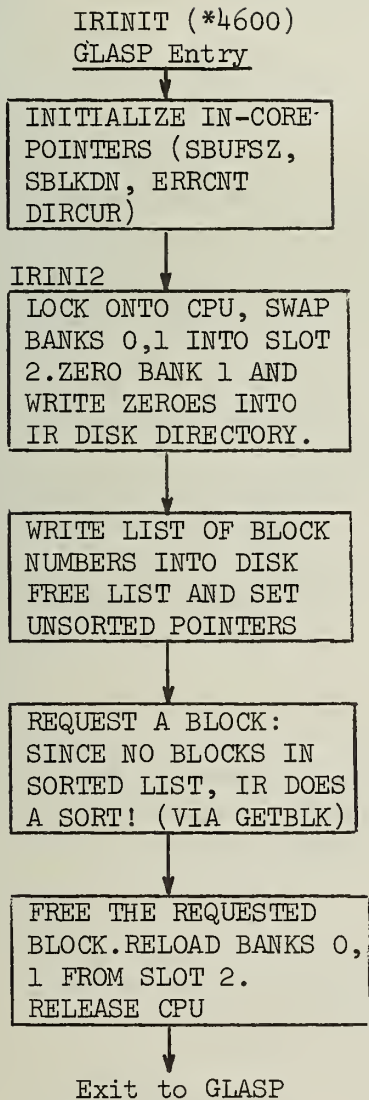


DEX1 (DEXCHK):33

DEXCHK (*4000)
Level 3 Entry



PSIR(IRINIT):37



U. S. ATOMIC ENERGY COMMISSION
UNIVERSITY-TYPE CONTRACTOR'S RECOMMENDATION FOR
DISPOSITION OF SCIENTIFIC AND TECHNICAL DOCUMENT

(See Instructions on Reverse Side)

1. AEC REPORT NO.

COO-1469-0190

2. TITLE

GRASS: System Software Description

3. TYPE OF DOCUMENT (Check one):

- ☒ a. Scientific and technical report
☐ b. Conference paper not to be published in a journal:

Title of conference _____

Date of conference _____

Exact location of conference _____

Sponsoring organization _____

- ☐ c. Other (Specify) _____

4. RECOMMENDED ANNOUNCEMENT AND DISTRIBUTION (Check one):

- ☒ a. AEC's normal announcement and distribution procedures may be followed.
☐ b. Make available only within AEC and to AEC contractors and other U.S. Government agencies and their contractors.
☐ c. Make no announcement or distribution.

5. REASON FOR RECOMMENDED RESTRICTIONS:

6. SUBMITTED BY: NAME AND POSITION (Please print or type)

C. W. Gear, Professor
and Principal Investigator

Organization

Department of Computer Science
University of Illinois
Urbana, Illinois 61801

Signature

Charles W. Gear

Date

August 1971

FOR AEC USE ONLY

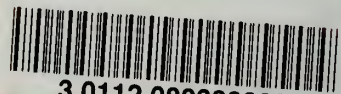
7. AEC CONTRACT ADMINISTRATOR'S COMMENTS, IF ANY, ON ABOVE ANNOUNCEMENT AND DISTRIBUTION RECOMMENDATION:

8. PATENT CLEARANCE:

- ☐ a. AEC patent clearance has been granted by responsible AEC patent group.
☐ b. Report has been sent to responsible AEC patent group for clearance.
☐ c. Patent clearance not required.



UNIVERSITY OF ILLINOIS-URBANA
510.84 IL6R no. C002 no. 463-468/1971
Control point design using modular logic



3 0112 088399883